

NECパーソナルコンピュータ
PC-9800シリーズ

NEC

Software Library

MS-DOS™3.3C
プログラマーズリファレンス
マニュアル Vol.2



Software Library

MS-DOS™3.3C
プログラマーズリファレンス
マニュアル Vol.2

ご注意

- (1) 本書の内容の一部又は全部を無断転載することは禁止されています。
- (2) 本書の内容に関しては将来予告なしに変更することがあります。
- (3) 本書は内容について万全を期して作成いたしましたが、万一御不審な点や誤り、記載もれなどお気づきのことがありましたら御連絡下さい。
- (4) 運用した結果の影響について(3)項にかかわらず責任を負いかねますので御了承下さい。

Microsoft (マイクロソフト) のロゴは米国マイクロソフト社の商標です。

MS-DOS は米国マイクロソフト社の商標です。

386, 386SXは米国インテル社の商標です。

Original Copyright © 1987 Lotus Development Corporation/Intel Corporation/Microsoft Corporation

Original Copyright © 1985, 1988, 1989 NEC Corporation

Copyright © 1988, 1989, 1990 NEC Corporation

Translation © 1988, 1989, 1990 マイクロソフト株式会社/NEC Corporation

輸出する際の注意事項

本製品 (ソフトウェア) は、外国為替および外国貿易管理法の規定により、戦略物資等輸出規制品に該当します。従って、日本国外に持出す際には日本国政府の輸出許可申請等必要な手続きをお取り下さい。

はじめに

本書「MS-DOS プログラマーズリファレンスマニュアル Vol.2」は、MS-DOS プログラマーズリファレンスマニュアル Vol.1 の続編として、標準的な MS-DOS のシステムコールからさらに拡張された機能や、いくつかの周辺装置を制御するデバイスドライバの技術情報を提供するものです。

□本書の構成

本書は 1 章から 6 章で構成されています。

第 1 章 本体機能に関する拡張機能呼び出しを解説しています。

第 2 章 日本語処理機能に関する拡張機能呼び出しを解説しています。これは、AI かな漢字変換用のデバイスドライバ(NECAIK1.DRV, NECAIK2.DRV)や文節変換用のデバイスドライバ(NECDIC.DRV)に関する技術情報です。

第 3 章 マウス用デバイスドライバ(MOUSE.SYS)に関する技術情報です。

第 4 章 グラフィックスドライバ(GRAPH.SYS)に関する技術情報です。

第 5 章 拡張メモリ(1 メガバイト以上のアドレス空間のメモリ)を制御する、EMS インターフェイスに関する技術情報です。

第 6 章 フォントドライバ FONT.SYS に関する技術情報です。

目次

第1章 本体機能

1.1 イントロダクション	1
1.2 拡張機能の利用方法	1
1.3 拡張機能呼び出し	1
RS-232C ポートの初期化	2
キーの取得	4
キーの設定	8
RS-232C ポートの操作	10
CTRL+FUNC	12
直接コンソール出力	13
プリンタモード	17

第2章 日本語処理

2.1 イントロダクション	19
2.2 使用方法	19
2.2.1 日本語入力用デバイスドライバの組み込み	19
2.2.2 ファンクションコールの使用法	20
2.2.3 AI かな漢字変換と EMS	21
2.3 日本語入力の拡張機能呼び出し	21
アプリケーションプログラムへの開放	23
アプリケーションプログラムからの使用禁止	24
キーボードからの日本語入力の禁止・許可	25
学習機能の有無	26
ローマ字列をカナ文字列に変換	27
1 バイト JIS 文字列を全角文字列に変換	28
1 バイト JIS 文字列を半角文字列に変換	29
辞書のオープン	30
辞書のクローズ	32

語句の登録	33
語句の削除	34
語句の学習	35
語句の変換（文節変換：最初の候補）	37
語句の変換（文節変換：次候補）	39
語句の変換（文節変換：前候補）	40
日本語入力モードに入る	41
日本語入力モードから抜ける	42
日本語入力のモードセット	43
日本語入力のモード取得	44
2バイト JIS をシフト JIS に変換	45
シフト JIS を2バイト JIS に変換	46
逐次／連文節変換ドライバの有無取得	47
辞書の先読み	50
連文節変換	56
次候補（連文節変換）	59
前候補（連文節変換）	62
学習（連文節）	64
先読み機能の有無	67

第3章 マウスインターフェイス

3.1 イントロダクション	69
3.2 マウス用デバイスドライバの組み込み	69
3.3 マウス用デバイスドライバについて	70
3.3.1 マウス用デバイスドライバの機能	70
3.3.2 ファンクションコールの方法	70
3.4 マウス用デバイスドライバのための予備知識	71
3.4.1 接続ディスプレイの種類と解像度	71
3.4.2 割り込みベクタ	71
3.4.3 割り込み周期	71
3.4.4 画面の座標系	71
3.4.5 表示画面	72
3.4.6 マウスカーソル	73
3.4.7 ミッキー	73

3.5 マウスファンクション	73
環境のチェック	74
カーソル表示	75
カーソル消去	76
カーソル位置の取得	77
カーソル位置の設定	78
左ボタンの押下情報の取得	79
左ボタンの解放情報の取得	80
右ボタンの押下情報の取得	81
右ボタンの解放情報の取得	82
カーソルの形の設定	83
マウスの移動距離の取得	85
ユーザー定義サブルーチンのコール条件の設定	86
ミッキー／ドット比の設定	88
水平方向のカーソル移動範囲の設定	89
垂直方向のカーソル移動範囲の設定	90
カーソルの表示画面の設定	91
グラフィック用 VRAM の設定と実装状況の取得	92
3.6 各パラメータの初期値	93

第4章 グラフィックスドライバ

4.1 イントロダクション	95
4.2 グラフィック用デバイスドライバ	95
4.2.1 デバイスドライバの組み込み	95
4.2.2 ファンクション一覧	96
4.3 ファンクションの呼び出し方法と使用例	97
4.3.1 ファンクションの呼び出し手順	97
4.3.2 マクロアセンブラでの使用例	100
4.3.3 C 言語での使用例	102
4.4 ファンクションプログラミングインターフェイス	103
グラフィックの開始	104
グラフィックの終了	105
仮想 VRAM の生成	106
表示モードの設定	108

描画プレーンの設定	110
表示プレーンの設定	112
パレットの設定	113
ビューポート領域の設定	116
フォアグラウンドカラーの設定	118
バックグラウンドカラーの設定	119
ボーダーカラーの設定	120
表示スイッチの設定	121
表示領域の設定	122
中断処理ルーチンの設定	123
画面消去	125
点の描画	126
線の描画	128
三角形の描画	130
長方形の描画	133
台形の描画	135
円の描画	137
楕円の描画	139
閉領域の塗りつぶし	141
グラフィックイメージの取得	143
グラフィックイメージの設定	145
領域転送	147
領域移動	149
バージョンの取得	150
プレーン数の取得	151
表示モードの取得	153
描画プレーンの取得	154
表示プレーンの取得	155
パレットの取得	156
ビューポート領域の取得	157
フォアグラウンドカラーの取得	158
バックグラウンドカラーの取得	159
ボーダーカラーの取得	160
表示スイッチの取得	161
指定座標のパレットの取得	162

表示領域の取得	163
中断処理ルーチンの取得	164
4.5 エラーコード一覧	165
4.6 専用高速描画版(GRP_H98. LIB)と GRAPH. LIB の互換性について	165

第5章 EMS インターフェイス

5.1 イントロダクション	167
5.2 拡張メモリを使用するプログラムの書き方	170
5.2.1 すべてのプログラムが実行しなければならないこと	170
5.2.2 応用プログラミング	171
5.2.3 物理ページのマッピングの状態を保存する	171
5.2.4 ハンドルの検索とページカウント	171
5.2.5 複数ページのマッピングとアンマッピング	171
5.2.6 ページのリアロケート	172
5.2.7 ハンドルの使用とハンドルへの名前の割り当て	172
5.2.8 ハンドルの属性の使用	172
5.2.9 ページマップの変更と飛び越し／呼び出し	173
5.2.10 メモリ領域の移動と交換	173
5.2.11 物理ページの数と各物理ページのアドレスを得る	173
5.2.12 オペレーティングシステムファンクション	173
5.3 プログラミングのガイドライン	175
5.4 EMS ファンクションリクエスト	177
5.4.1 ファンクションリクエスト一覧	177
5.4.2 EMS ファンクションリクエスト	180
Get Status	181
Get Page Frame Address	182
Get Unallocated Page Count	183
Allocate Pages	184
Map/Unmap Handle Page	187
Deallocate Pages	189
Get Version	191
Save Page Map	192
Restore Page Map	194
Reserved	196
Get Handle Count	197

Get Handle Pages	198
Get All Handle Pages	199
Get Page Map	201
Set Page Map	202
Get & Set Page Map	203
Get Size of Page Map Save Array	205
Get Partial Page Map	206
Set Partial Page Map	208
Get Size of Partial Page Map Save Array	209
Map/Unmap Multiple Handle Pages	210
Map/Unmap Multiple Handle Pages	212
Reallocate Pages	215
Get Handle Attribute	218
Set Handle Attribute	220
Get Attribute Capability	222
Get Handle Name	223
Set Handle Name	225
Get Handle Directory	227
Search for Named Handle	229
Get Total Handles	230
Alter Page Map & Jump	231
Alter Page Map & Call	234
Get Page Map Stack Space Size	238
Move Memory Region	239
Exchange Memory Region	244
Get Mappable Physical Address Array	249
Get Mappable Physical Address Array Entries	251
Get Hardware Configuration Array	252
Get Unallocated Raw Page Count	255
Allocate Standard Pages	257
Allocate Raw Pages	260
Alternate Map Register Set	264
Get Alternate Map Register Set	266
Set Alternate Map Register Set	269
Get Alternate Map Save Array Size	272

Allocate Alternate Map Register Set	273
Deallocate Alternate Map Register Set	275
Allocate DMA Register Set	277
Enable DMA on Alternate Map Register Set	279
Disable DMA on Alternate Map Register Set	281
Deallocate DMA Register Set	283
Prepare Expanded Memory Hardware For Warm Boot	284
Enable OS/E Function Set	285
Disable OS/E Function Set	288
Return Access Key	290
Get Page frame Status	295
Enable/Disable Page frame	296
5.5 ファンクションとステータスコードのクロスリファレンス	297
5.6 拡張メモリマネージャの有無のテスト	305
5.6.1 プログラムでどちらの方法を用いるとよいか	305
5.6.2 "open handle" 法	305
5.6.3 "get interrupt vector" 法	309
5.7 拡張メモリマネージャのインプリメンテーションへのガイドライン	311
5.8 ファンクション 28 におけるオペレーティングシステムの利用	313
5.9 用語	315

第 6 章 フォントドライバ

6.1 イントロダクション	319
6.2 フォントドライバ	320
6.2.1 フォントドライバの組み込み	320
6.2.2 ファンクション一覧	321
6.3 フォントドライバの呼び出し方法と使用例	321
6.3.1 ファンクションの呼び出し手順	321
6.3.2 マクロアセンブラでの使用例	322
6.3.3 文字フォントの利用例	324
6.4 ファンクションプログラミングインターフェイス	324
バージョンの取得	325
フォント情報の設定	326
フォント情報の取得	331

フォントの取得	332
動作指定フラグに 0 が指定された場合	332
動作指定フラグに 1 が指定された場合	334
動作指定フラグに 2 が指定された場合	336
6.5 エラーコード一覧	337
索 引	339

第1章

本体機能

1.1 イン트로ダクション

PC-9800 シリーズの本体ハードウェア資源を、有効かつ効率的に使用するために、いくつかの拡張機能が用意され、プログラムで利用できるようになっています。

ここでは、これらの拡張機能を解説します。なお、PC-9800 シリーズで、ノーマルモードとハイレゾリレーションモードで動作や操作に違いがある場合は、原則としてノーマルモードでの解説を基にして、ハイレゾリレーションモードにおける違いを解説します。

1.2 拡張機能の利用方法

拡張機能を呼び出す（コールする）ときは、CLレジスタに機能コードを格納し、その他の必要な情報を各レジスタに設定して、割り込みタイプ DCH（INT DCH）を実行します。

呼び出し後のレジスタの内容は、リターンで定義されているレジスタ以外はすべて保障されます。機能が定義されていない機能コードによる呼び出しでは、何も実行されません。

1.3 拡張機能呼び出し

PC-9800 シリーズでは、次のような機能が拡張機能として用意されています。

機能コード（16 進）	機能
0AH	RS-232C ポートの初期化
0CH	キーの取得
0DH	キーの設定
0EH	RS-232C ポートの操作
0FH	CTRL+FUNC
10H	直接コンソール出力
11H	プリンタモード

ここでは、各拡張機能呼び出しごとに解説を行います。

RS-232C ポートの初期化

機能コード 0AH

機能 指定された Ch No. の RS-232C ポートを初期設定する(ハイレゾリューションモードのみ)。

コール CL = 0AH
DX = パラメータ

DH =	データ (ビットの位置)								機 能	
	7	6	5	4	3	2	1	0		
								0 1	X パラメータ	無効 有効
								0		
					1 1	0 1			データビット長	7ビット 8ビット
				0 1					パリティチェック	なし あり
			0 1						パリティ指定	奇数 偶数
	0 1	1 1							ストップビット長	1ビット 2ビット
DL =	7	6	5	4	3	2	1	0		
					0 0 0 0 0 0 0 1	0 0 1 0 1 0 1 0	0 0 1 1 0 1 1 1	0 1 1 1 0 0 0	ボーレート	無効 75ボー 150ボー 300ボー 600ボー 1200ボー 2400ボー 4800ボー 9600ボー
	0 0 0	0 0 0	0 0 1	0 1 0						Ch. No.
										0
										1
										2

リターン AX = 0 正常終了
AX = FFFFH 異常終了

解説 レジスタDXにセットされたパラメータでRS-232Cポートが初期設定されます。
この機能は、PC-98XL/XAなどのハイレゾリューションモードでのみ使用可能です。

機能コード 0EH は、ノーマルモード、ハイレゾリューションモード共通に使用できます。

Ch No.1 および 2 については、拡張ボードがセットされていなければ、初期化は行われません。また、レジスタ DL でセットするボーレートは Ch No.0 についてのみ有効です。Ch No.1 および 2 は拡張ボード上のスイッチにより $\times 16$ モードでセットしてください。

キーの取得

機能コード 0CH

機能 ファンクションキーやカーソル移動キーなどの取得

コール CL = 0CH

DX = データバッファのオフセット

DS = データバッファのセグメント

AX = 0000H	ノーマルモード全ソフトキーを取得
00FFH	ハイレゾリューションモード全ソフトキーを取得
0001~000AH	f・1~f・10 の取得
000B~0014H	SHIFT+f・1~SHIFT+f・10 の取得
0015~001FH	カーソル移動キーなど*1の取得
0020~0024H	f・11~f・15 の取得
0025~0029H	SHIFT+f・11~SHIFT+f・15 の取得
002A~0038H	CTRL+f・1~CTRL+f・15 の取得
0100H	データキー割り当てバッファの内容の取得
0101H	データキー割り当てバッファの残りサイズの取得

AX=0101H のときは、レジスタ DS, DX にバッファアドレスをセットする必要はありません。

解説 ファンクションキーやカーソル移動キーなどの取得機能は、レジスタ DX にアドレスが格納されているバッファに、ファンクションキーやカーソル移動キーに現在割り当てられている機能を書き込みます。

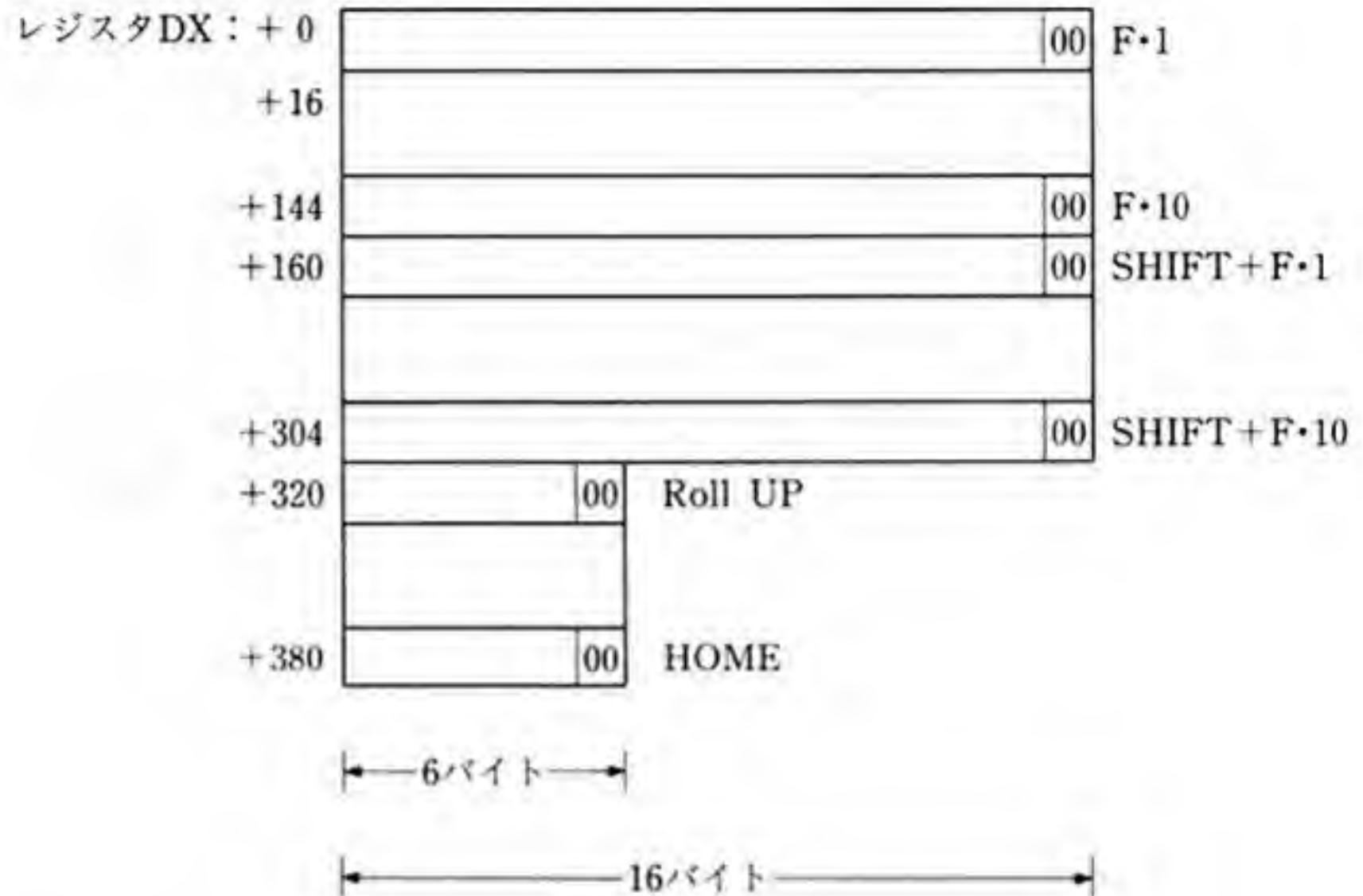
レジスタ AX に 0100H をセットしてこの機能呼び出しを行うと、レジスタ DX に格納されているバッファに、データキーに現在再割り当てされている機能を書き込みます。また、レジスタ AX に 0101H をセットしてこの機能呼び出しを行うと、データキーに機能を再割り当てするための内部バッファの残りバイト数をレジスタ AX に返します。

* 1

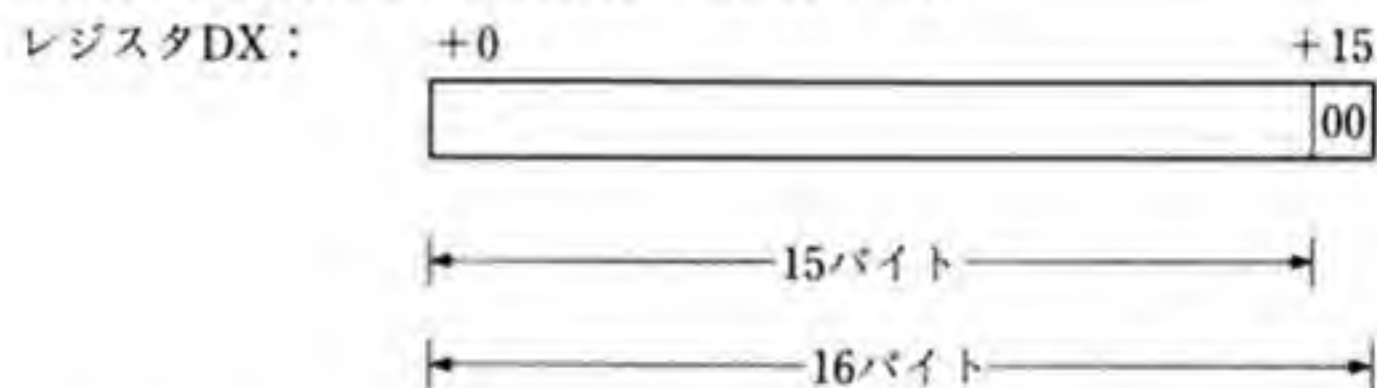
AX= 0015H	ROLLUP	0016H	ROLLDOWN
0017H	INS	0018H	DEL
0019H	↑	001AH	←
001BH	→	001CH	↓
001DH	ノーマルモード時 HOMECLR ハイレゾリユーションモード時 CLR		
001EH	HELP		
001FH	ノーマルモード時 SHIFT+HOMECLR ハイレゾリユーションモード時 HOME		

バッファの形式：

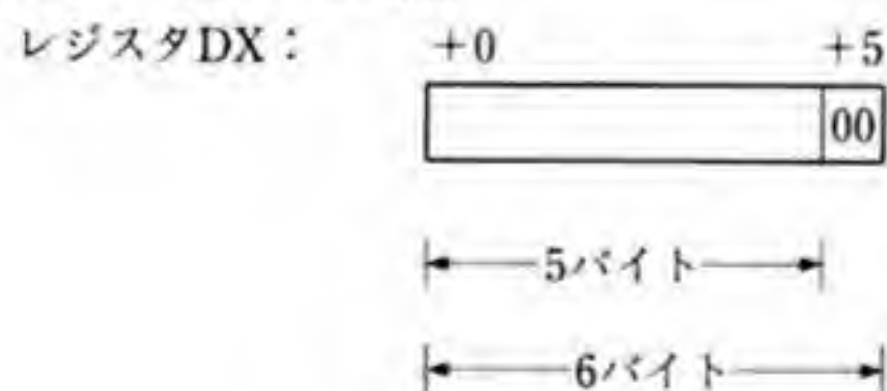
AX=0000Hの場合



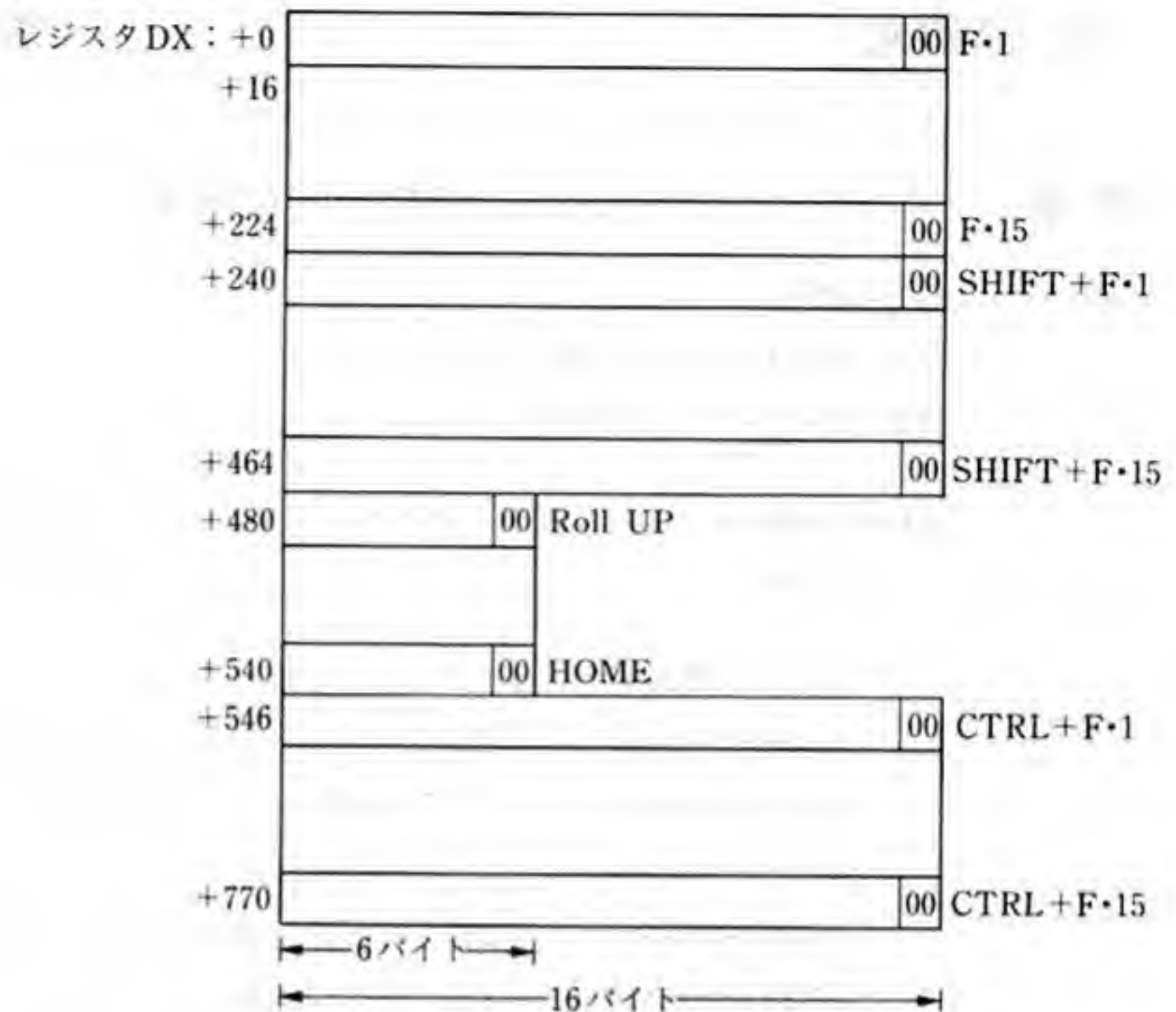
AX=0001H～0014Hあるいは0020H～0038Hの場合



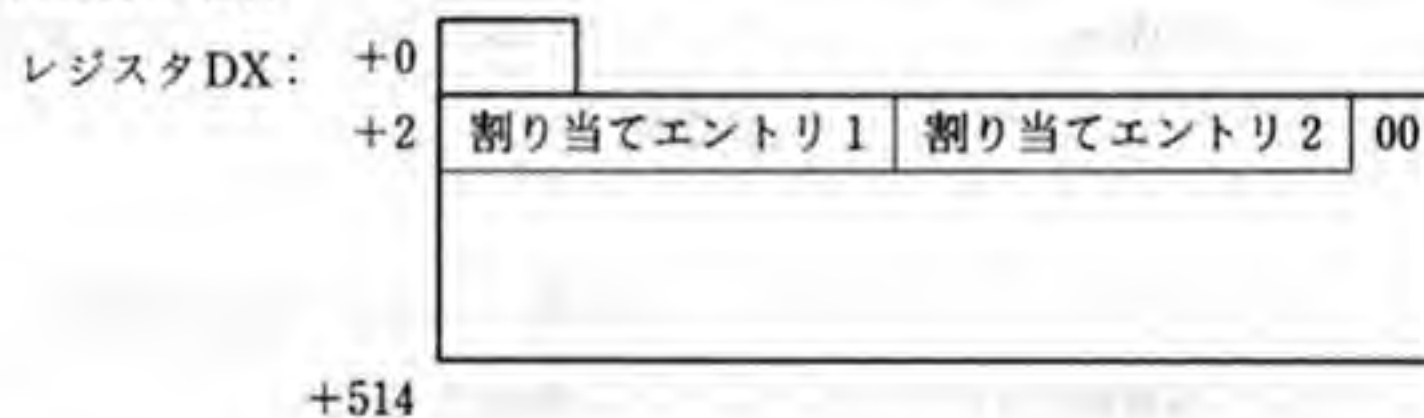
AX=0015H～001FHの場合



AX=00FFHの場合



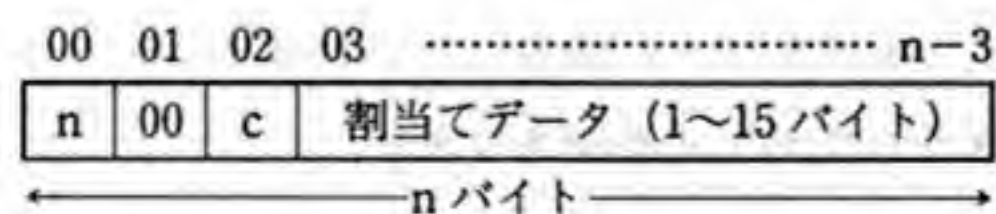
AX=0100Hの場合



オフセット 00 から 1 ワードには、割り当てエントリ数が格納されています。

オフセット 02 から 512 バイトはバッファとして使用され、割り当てエントリが格納されます。

1つの割り当てエントリは以下の形式になっています。



n割り当てエントリ全体の長さ (バイト数)

c割り当てるキーコード (英数, 英記号, カナ, カナ記号
20H~7EH, A0H~DFH)

キーの設定

機能コード0DH

機能 ファンクションキーやカーソル移動キーなどの設定

コール CL = 0DH

DX = データバッファのオフセット

DS = データバッファのセグメント

AX = 0000H	ノーマルモード全ソフトキーを設定
00FFH	ハイレゾリューションモード全ソフトキーを設定
0001~000AH	f・1~f・10 の設定
000B~0014H	SHIFT+f・1~SHIFT+f・10 の設定
0015~001FH	カーソル移動キーなど*1の設定
0020~0024H	f・11~f・15 の設定
0025~0029H	SHIFT+f・11~SHIFT+f・15 の設定
002A~0038H	CTRL+f・1~CTRL+f・15 の設定
0100H	データキー割り当てバッファの内容設定*2
0101H	データキー割り当てバッファに割り当てエントリを1つ追加

解説 ファンクションキーやカーソル移動キーなどの設定機能は、レジスタ DX にアドレスが格納されているバッファの機能をファンクションキーやカーソル移動キーに設定します。

レジスタ AX=0000H のときは、ノーマルモードで使用可能なファンクションキー、カーソル移動キーすべてに機能を設定します。

レジスタ AX=00FFH のときは、ハイレゾリューションモードで使用可能なファンクションキー、カーソル移動キーすべてに機能を設定します。

データバッファの形式は、機能コード 0CH と同じです。

各キーに対する有効文字列の最後には、00H が 16 バイトあるいは 6 バイトを満たすまでおぎなっておく必要があります。

レジスタ AX に 0100H をセットしてこの機能呼び出しを行うと、レジスタ DX にアドレスが格納されているバッファのデータキー割り当て情報をシステムの内部バッファに設定します。

また、レジスタ AX に 0101H をセットしてこの機能呼び出しを行うと、内部バッファにデータキーへの機能割り当てを 1 エントリだけ追加します。

割り当てエントリの形式は機能コード 0CH を参照してください。

* 1

AX=0015H	ROLLUP	0016H	ROLLDOWN
0017H	INS	0018H	DEL
0019H	↑	001AH	←
001BH	→	001CH	↓
001DH	ノーマルモード時 HOMECLR		
	ハイレゾリョーションモード時 CLR		
001EH	HELP		
001FH	ノーマルモード時 SHIFT+HOMECLR		
	ハイレゾリョーションモード時 HOME		

* 2

データキー割り当てバッファの内容は、機能コード 0CH を参照してください。

このコールではオフセット 0 からの 1 ワードにエントリ数を格納する必要はありません。

注意： CTRL+f・1~CTRL+f・15に割り当てた機能を実際に使用するためには、機能コード 0FH AX=0000H の呼出しによって CTRL+f・1~CTRL+f・15 をソフトキー化しなければなりません。

RS-232C ポートの操作

機能コード 0EH

機能 指定された ch No. の RS-232C ポートの初期設定または受信データ長の通知

コール CL = 0EH

DL = パラメータ

00H チャンネル 0 の受信データ通知

10H チャンネル 1 の受信データ通知

20H チャンネル 2 の受信データ通知

01H チャンネル 0 の初期設定

11H チャンネル 1 の初期設定

21H チャンネル 2 の初期設定

DL = 01H, 11H, 21H (初期設定) のとき

BX = パラメータ

BH =	データ (ビット位置)								機 能	
	7	6	5	4	3	2	1	0		
								0 1	X パラメータ	無効 有効
								0		
					1 1	0 1			データビット長	7ビット 8ビット
						0 1			パリティチェック	なし あり
							0 1		パリティ指定	奇数 偶数
	0 1	1 1							ストップビット長	1ビット 2ビット

BL =	データ (ビット位置)								機 能	
	7	6	5	4	3	2	1	0		
					0 0 0 0 0 0 0 0 1	0 0 0 0 1 1 1 1 0	0 0 1 1 0 0 1 1 0	0 1 1 1 0 0	ボーレート	無効 75 ボー 150 ボー 300 ボー 600 ボー 1200 ボー 2400 ボー 4800 ボー 9600 ボー
	0 0 0 0									

リターン

DL=00H, 10H, 20H (データ長取得) のとき

AX=受信データのデータ長

DL=01H, 11H, 21H (初期設定) のとき

AX=0000H 正常終了

FFFFH 異常終了 (拡張 RS-232C ボードが実装されていません)

解 説

DL=x0H (x は 0~2) のとき、指定された RS-232C ポートに受信しているデータ長を、レジスタ AX に返します。

DL=x1H (x は 0~2) のとき、レジスタ BX にセットされたパラメータで RS-232C ポートが初期設定されます。

ch No.1 および 2 については、拡張 RS-232C ボードが本体に実装されていなければ初期設定は行われません。また、レジスタ BL にセットされるボーレートは Ch No.0 のみ有効です。Ch No.1 および 2 は拡張 RS-232C ボード上のスイッチにより×16 モードでセットしてください。

CTRL+FUNC

機能コード 0FH

機能 CTRL+ファンクションキーのソフトキー化およびソフトキー解除

コール CL = 0FH

AX = 0000H CTRL+ファンクションキーのソフトキー化

0001H CTRL+ファンクションキーのソフトキー解除

解説 ノーマルモードの **CTRL** + **f.1** ~ **f.10** , ハイレゾリューションモードでの **CTRL** + **f.1** ~ **f.15** は、通常の MS-DOS では DIRECT CONSOLE I/O でそのキー値を得ることができません。しかし、レジスタ AX に 0000H をセットし、この機能呼び出しを行うことで、DIRECT CONSOLE I/O でキー値を得ることができるようになります。

CTRL + **f.1** ~ **f.10** または **CTRL** + **f.1** ~ **f.15** の扱いを通常の MS-DOS のものに戻すには、レジスタ AX に 0001H をセットしてこの機能呼び出しを行ってください。

KEY コマンドで設定した CTRL+ファンクションキーの機能を利用するためには、レジスタ AX に 0000H を入れて、この機能呼び出しを行わなければなりません。

直接コンソール出力

機能コード 10H

機 能 ディスプレイ画面への直接出力

コール CL = 10H

AH = サブファンクション番号 (00-0EH)

その他 = サブファンクションごとに必要なレジスタ

解 説 AH にセットされたサブファンクション番号に応じて、ディスプレイ画面に対して直接出力動作を行います。サブファンクションごとの機能およびコール条件は次のとおりです。

AH = 00H

機 能 : ディスプレイ画面に、1 バイトのデータを出力します。漢字を出力するにはシフト JIS コードの第 1 バイト、第 2 バイトを順に出力してください。

コール : DL = 出力するキャラクタ

AH = 01H

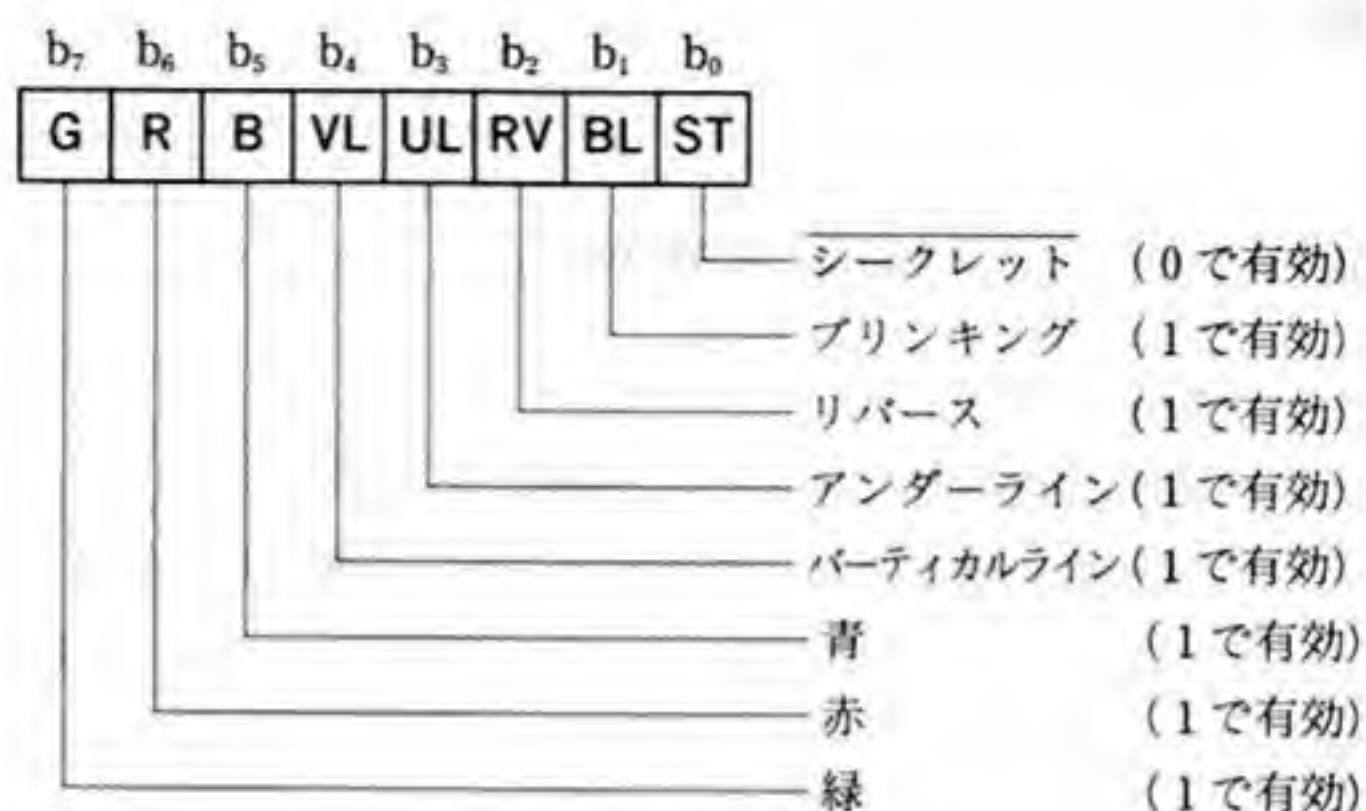
機 能 : ディスプレイ画面に文字列を出力します。文字列の終わりには '\$' をセットしてください。

コール : DS : DX = 文字列の先頭アドレス

AH = 02H

機 能 : 文字の属性を変更します。このサブファンクション実行後は、設定した属性が以後に続く文字に対して適用され、次の変更まで有効です。

コール：DL=文字の属性



AH = 03H

機能：カーソル位置の設定を行います。

コール：DH=ライン

DL=カラム

AH = 04H

機能：カーソルを同じカラムで下に1行移動します。カーソルが最終行にある場合は1行スクロールアップします。

コール：なし

AH = 05H

機能：カーソルを同じカラムで上に1行移動します。カーソルが先頭行にある場合は1行スクロールダウンします。

コール：なし

AH = 06H

機能：カーソルを同じカラムで上にn行移動します。カーソルが先頭行にあるか、先頭行を越えた場合には、先頭行に位置します。n=0の場合にはn=1として処理します。

コール：DL=移動行数 n

AH = 07H

機能：カーソルを同じカラムで下にn行移動します。カーソルが最終行にあるか、最終行を越えた場合には、最終行に位置します。n=0の場合にはn=1として処理します。

コール：DL=移動行数 n

AH = 08H

機 能：カーソルを右に n カラム移動します。カーソルが行の右端にあるか、右端を越えた場合には右端に位置します。n=0 の場合には n=1 として処理します。

コール：DL=移動カラム数 n

AH = 09H

機 能：カーソルを左に n カラム移動します。カーソルが行の左端にあるか、左端を越えた場合には左端に位置します。
n=0 の場合には n=1 として処理します。

コール：DL=移動カラム数 n

AH = 0AH

機 能：ディスプレイ画面のクリアをコール条件にしたがって行います。

コール：DL=0 カーソル位置から最終行右端までをクリアします。
 =1 先頭行左端からカーソル位置までをクリアします。
 =2 画面全体をクリアします。
 これらの値以外は無視されます。

AH = 0BH

機 能：現在行のクリアをコール条件にしたがって行います。

コール：DL=0 カーソル位置から行の右端までをクリアします。
 =1 行の左端からカーソル位置までをクリアします。
 =2 カーソルの位置する行を左端から右端までクリアします。

AH = 0CH

機 能：カーソルの位置する行以降を n 行下に移動し、空白の n 行を挿入します。カーソルは先頭の挿入行の左端に位置します。挿入行が最終行を越えた場合、移動する行が最終行を越えた場合は、その越えた行は失われます。

n=0 の場合は n=1 として処理します。

コール：DL=挿入する行数 n

AH = 0DH

機 能：カーソルの位置する行から下に n 行削除し、以降の行を上詰めます。カーソルの位置は詰められた行の左端になります。最終行を越えての削除は行われません。

n=0 の場合は n=1 として処理します。

コール：DL=削除する行数 n

AH = 0EH

機 能：81～9FH あるいは E0～FCH までのコードをシフト JIS コードの第1バイトとして扱うか、グラフ文字として扱うかのモード指定を行います。

コール：DL=0 シフト JIS モード（システムの既定値です）

=3 グラフ文字モード

上記の値以外は無視されます。

プリンタモード

機能コード 11H

機 能 プリンタモードを変更する

コール CL = 11H

AX = 0000H ドットスペイシングを行わないモードとなる

0001H ドットスペイシングを行うモードとなる

0020H 偶数回目のCTRL+PでプリンタにCR/LFコードを出力しない

0021H 偶数回目のCTRL+PでプリンタにCR/LFコードを出力する

解 説 レジスタ AX に 0000H をセットしてこの機能呼び出しを行うと、日本語プリンタにおいて、ANK(1バイト系英数カナ)文字と漢字の比率が1:1.5になります。
また、レジスタ AX に 0001H をセットしてこの機能呼び出しを行うと、日本語プリンタにおいて、ANK 文字と漢字の比率が1:2になります。

レジスタ AX に 0020H をセットしてこの機能呼び出しを行うと、偶数回目の **CTRL** + **P** 押下（画面出力のプリンタへのエコー解除）時に、プリンタに CR/LF コードを出力しないモードとなります。また、レジスタ AX に 0021H をセットしてこの機能呼び出しを行うと、偶数回目の **CTRL** + **P** 押下時にプリンタに CR/LF コードを出力するようになります。

第2章

日本語処理

2.1 イントロダクション

MS-DOS 3.3C では、スムーズに日本語(漢字やひらがななどの2バイトコード文字)を入力するための拡張機能が用意されています。この拡張機能はデバイスドライバの形で提供され、ユーザーがプログラム内で使用することができます。

提供される日本語入力用デバイスドライバには、次の2種類があります。

ファイル名	機 能
NECAIK1.DRV NECAIK2.DRV	AI かな漢字変換による日本語入力用デバイスドライバ
NECDIC.DRV	
	文節変換による日本語入力用デバイスドライバ

2.2 使用方法

ここでは、日本語入力用デバイスドライバの組み込み方法と、ファンクションコールの使用方法を解説します。

2.2.1 日本語入力用デバイスドライバの組み込み

日本語処理機能を利用するには、CONFIG.SYS ファイルに次のような行を追加して、日本語入力用デバイスドライバを組み込みます。

- AI かな漢字変換を利用する場合

DEVICE=NECAIK1.DRV

DEVICE=NECAIK2.DRV [/T] [/R] <辞書ファイル名>

<辞書ファイル名> を省略すると、カレントドライブのルートディレクトリのファイル名 "NECAI.SYS" となります。

/T スイッチは、AI 変換機能を使用しないで、逐次/連文節変換モードで使用する場合に指定します。このスイッチを省略すると、AI 逐次または AI 連文節変換モードになります。

/R スイッチは、連文節変換モードを使用する場合に指定します。このスイッチを省略すると、逐次変換モードになります。

<辞書ファイル名> を省略すると、カレントドライブのルートディレクトリのファイル名

“NECAI.SYS”（連文節変換モードでは“NECDIC.SYS”）になります。

●文節変換を利用する場合

DEVICE = NECDIC.DRV 〈辞書ファイル名〉

〈辞書ファイル名〉を省略すると、カレントドライブのルートディレクトリのファイル名“NECDIC.SYS”となります。

日本語入力用デバイスドライバは、キャラクタ系デバイスドライバですので、ADDDRV／DELDRV コマンドによって、MS-DOS の起動後に組み込み／削除することもできます（MS-DOS 3.3C ユーザーズリファレンスマニュアル参照）。

2.2.2 ファンクションコールの使用方法

日本語入力機能は、ユーザープログラム内で使用することができます。プログラムでこの機能を利用するは、INT DCHを用いてシステムコールを行います。

日本語入力の拡張機能呼び出す（コールする）には、まずCLレジスタに機能コードの番号を入れます。また、機能コード以外に必要な情報がある場合は、指定されているレジスタにその情報をすべて格納してから、割り込みタイプDCH（INT DCH）を実行します。

例）学習機能を無しに設定する

```
MOV     CL,227 ; CL=機能コード
MOV     AX,0   ; 学習機能無し
                ; そのほかに、必要な情報があれば各レジスタにセットする
INT     DCH
```

呼び出し後のレジスタの内容はリターンで定義されているレジスタ以外はすべて呼び出し前のレジスタの値を保持しています。なお、定義されていない機能コードをCLレジスタに入れ、呼び出しを行った場合は何も実行されません。

エラーコードが通知されるのは、異常終了（キャリーフラグがセットされている状態）の場合で、正常終了の場合、AXレジスタの値は不定です。

変換を行うとき、“読み”の中の次の文字や記号は変換されません。

カタカナ （読みの中のカタカナの部分）
記号類 — 「 」 、 。 ・

2.2.3 AI かな漢字変換と EMS

AI かな漢字変換ドライバ (NECAIK1.DRV, NECAIK2.DRV) は、通常約 128 K バイトのメインメモリ (主記憶) を使用しますが、EMS ドライバ (第 5 章参照) が組み込まれている場合は、AI かな漢字変換ドライバの一部分を拡張メモリに置くことができます。これによって、約 60 K バイト、アプリケーションプログラムで利用できるメインメモリ量が増加します。

このような拡張メモリの使い方をする場合は、CONFIG.SYS ファイル中で、EMS ドライバを指定してから、AI かな漢字変換ドライバを指定してください。

注意：ページフレームは、64 K バイト確保できなければなりません。

2.3 日本語入力の拡張機能呼び出し

日本語入力のために、次のような拡張機能呼び出しが用意されています。呼び出し方法については「2.2.2 ファンクションコールの使用方法」を参照してください。

機能コード	機能
10 進 16 進	
224 E0H	アプリケーションプログラムへの解放
225 E1H	アプリケーションプログラムからの使用禁止
226 E2H	キーボードからの日本語入力の禁止/許可
227 E3H	学習機能の有無
228 E4H	ローマ字をカナ文字に変換
229 E5H	1 バイト JIS 文字列を全角文字列に変換
230 E6H	1 バイト JIS 文字列を半角文字列に変換
231 E7H	辞書のオープン
232 E8H	辞書のクローズ
233 E9H	語句の登録
234 EAH	語句の削除
235 EBH	語句の学習
236 ECH	語句の変換 (文節変換：最初の候補)
237 EDH	語句の変換 (文節変換：次候補)
238 EEH	語句の変換 (文節変換：前候補)
239 EFH	日本語入力モードに入る
240 F0H	日本語入力モードから抜ける
241 F1H	日本語入力モードのセット
242 F2H	日本語入力モードの取得

243	F3H	2バイト JIS をシフト JIS に変換
244	F4H	シフト JIS を2バイト JIS に変換
247	F7H	逐次変換ドライバの有無の取得
248	F8H	辞書の先読みと逐次変換
249	F9H	連文節変換（最初の候補）
250	FAH	連文節変換（次候補）
251	FBH	連文節変換（前候補）
252	FCH	学習（連文節）
253	FDH	先読み機能の有無

ここでは、これらの機能ごとに解説を行います。

各機能のタイトルに、ついているマークは次の意味を表わします。

- (逐) AI かな漢字変換ドライバの、AI 逐次／逐次変換モードで利用できる機能。
- (連) AI かな漢字変換ドライバの、AI 連文節／連文節変換モードで利用できる機能。
- (文) 文節変換ドライバで利用できる機能

上記のマークのついていない機能は、AI 逐次変換、逐次変換、AI 連文節変換、連文節変換、文節変換のいずれでも利用できる機能です。

アプリケーションプログラムへの開放

機能コード 224

機 能 日本語入力機能をアプリケーションプログラムに開放

コール AX=0

リターン AX=1 日本語入力機能を使用できます。
0 日本語入力機能を使用できません。

解 説 日本語入力機能をアプリケーションプログラムに開放します。
また、日本語入力機能がシステムに組み込まれているかどうかの情報を返します。
アプリケーションプログラムインターフェースを使用するときには、最初に必ずこのコールを実行しなければなりません。
AI 逐次/AI 連文節, 逐次/連文節の機能を使用する場合は、このコールの後に必ず機能コード 247（逐次変換ドライバの有無取得）をコールしてください。

アプリケーションプログラムからの使用禁止 機能コード 225

機 能	アプリケーションプログラムからの日本語入力機能の使用を禁止
-----	-------------------------------

コール なし

リターン キャリーフラグがセットされた場合

AX = 1 日本語入力機能がアプリケーションプログラムに開放されて
いません。

キャリーフラグがセットされない場合

正しく終了しました。

解 説 日本語入力機能をアプリケーションプログラムから使用できないようにします。
また、これをコールすると、現在オープン中の辞書はクローズされます。

アプリケーションプログラムから日本語入力機能を使用した場合には、最後に必ずこのコールを実行しなければなりません。

キーボードからの日本語入力の禁止・許可

機能コード 226

機能

キーボードからの日本語入力の禁止および許可

コール

AX = 0 許可
 0 以外 禁止

リターン

キャリーフラグがセットされた場合

AX = 1 日本語入力機能がアプリケーションプログラムに開放されてい
 ません。

キャリーフラグがセットされていない場合

正しく終了しました。

解説

キーボードからの日本語入力を、AX レジスタの値により禁止または許可しま
 す。

機能コード 225（アプリケーションプログラムからの日本語入力機能の使用を
 禁止）をコールすると、キーボードからの日本語入力は許可状態になります。

学習機能の有無

機能コード 227

機能	学習機能の有無を設定
----	------------

コール	AX=0 学習機能無し 0以外 学習機能有り
-----	-----------------------------------

リターン	キャリーフラグがセットされた場合 AX=1 日本語入力機能がアプリケーションプログラムに開放されていません。 キャリーフラグがセットされない場合 正しく終了しました。 AX=0 設定前は学習機能無しでした。 1 設定前は学習機能有りでした。
------	--

解説	学習機能の有無を設定します。 語句の学習を行うときには、このコールで“学習機能有り”がセットされていなければなりません。
----	---

ローマ字列をカナ文字列に変換

機能コード 228

機 能 ローマ字列（1 バイト JIS）をカナ文字列（1 バイト JIS）に変換

コール DS:SI = ローマ字列へのポインタ（終端は NULL）
 ES:DI = 変換結果格納域へのポインタ
 AX = 変換結果格納域の大きさ（バイト数）

リターン キャリーフラグがセットされた場合
 AX=1 日本語入力機能がアプリケーションプログラムに開放されてい
 ません。
 キャリーフラグがセットされない場合
 正しく終了しました。
 AX=変換結果の長さ（バイト数）
 CX=変換されたローマ字列の長さ（バイト数）

解 説 与えられたローマ字列（1 バイト JIS）をカナ文字列（1 バイト JIS）に変換し
 ます。
 変換は、変換結果が変換結果格納域に納まる範囲内で行われます。
 また、ローマ字列内に、ローマ字規則（日本語入力ガイドの付録を参照してく
 ださい）に反する文字例および英数字以外の文字が発見された場合には、この場
 で変換を終了します。
 ローマ字列の最後に *N* があつた場合は変換されません。*ン* と変換したい
 ときには *N' * と格納してください。
 ローマ字列の最後には NULL (00H) をセットしておいてください。

1 バイト JIS 文字列を全角文字列に変換

機能コード 229

機能 1 バイト JIS 文字列を全角文字列（シフト JIS）に変換

コール DS:SI=1 バイト JIS 文字列へのポインタ（終端は NULL）

ES:DI=変換結果格納域へのポインタ

AX =変換結果格納域の大きさ（バイト数）

DX =0 カナを全角カタカナに変換

1 カナを全角ひらがなに変換

リターン キャリーフラグがセットされた場合

AX=1 日本語入力機能がアプリケーションプログラムに開放されていません。

キャリーフラグがセットされない場合

正しく終了しました。

AX=変換結果の長さ（バイト数）

CX=変換された 1 バイト JIS 文字列の長さ（バイト数）

解説 与えられた 1 バイト JIS 文字列を全角文字列（シフト JIS）に変換します。

変換は、変換結果が、変換結果格納域に納まる範囲内で行われます。

また、1 バイト JIS 文字列内に 1 バイト JIS 以外の文字（制御コード（00H～1FH）と漢字（80H～9FH, E0H～FCH））が発見された場合には、その場で変換を終了します。

1 バイト JIS 文字列の最後には NULL（00H）をセットしてください。

濁点、半濁点は直前の文字と合わせて処理されます（カ・→が、ハ・→ぱなど）。

変換結果の格納形式は上位バイト・下位バイトの順です（A（8260H）は、82H, 60H と格納されます）。

1 バイト JIS 文字列を半角文字列に変換

機能コード 230

機能 1 バイト JIS 文字列 (1 バイト JIS) を半角文字列 (シフト JIS) に変換

コール DS:SI=1 バイトコード文字列へのポインタ (終端は NULL)

ES:DI=変換結果格納域へのポインタ

AX = 変換結果格納域の大きさ (バイト数)

リターン キャリーフラグがセットされた場合

AX=1 日本語入力機能がアプリケーションプログラムに開放されていません。

キャリーフラグがセットされない場合

正しく終了しました。

AX=変換結果の長さ (バイト数)

CX=変換された 1 バイト JIS 文字列の長さ (バイト数)

解説 与えられた 1 バイト JIS 文字を半角文字列 (シフト JIS) に変換します。

変換は、変換結果が変換結果格納域に納まる範囲内で行われます。

また、1 バイト JIS 文字列内に 1 バイト JIS 以外の文字 (制御コード (00H ~ 1FH) と、漢字 (80H ~ 9FH, E0H ~ FCH)) が発見された場合には、その場で変換を終了します。

1 バイト JIS 文字列の最後には NULL (00H) をセットしてください。

空白 (20H) は変換されず、20H のままです。変換結果の格納形式は上位バイト・下位バイトの順です (A (8260H) は 82H, 60H と格納されます)。

辞書のオープン

機能コード 231

機能 辞書のオープン

コール DS:SI=辞書ファイル名へのポインタ
ES:DI=12 バイトの領域をさすポインタ

リターン キャリーフラグがセットされた場合

AX=1 日本語入力機能がアプリケーションプログラムに開放されていません。

3 既に辞書がオープンされています。

4 指定された辞書が見つかりません。

14 ディスクの読み込み中にエラーが発生しました。

キャリーフラグがセットされない場合

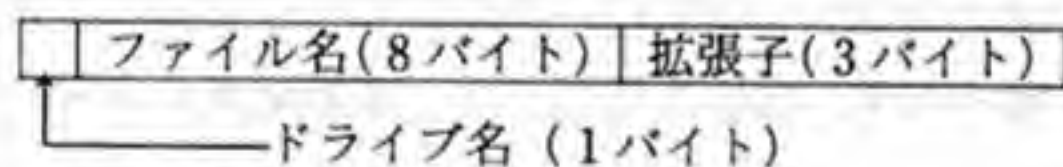
正しく終了しました。

ES:DI=直前の辞書情報へのポインタ

これは、オープンが成功しても失敗してもセットされます。

解説 指定された辞書をオープンします。

DS:SI には辞書ファイル名へのポインタをセットしておきます。
ファイル名の形式はつぎの形式になっていなければなりません。



ドライブ名: 0 カレントドライブ

1 ドライブ A:

2 ドライブ B:

⋮

ファイル名: 8 文字までの長さのファイル名を大文字でセットします。ファイル名はフィールドの先頭からセットし 8 文字に満たない場合には残りにスペース (20H) をセットしてください。
ファイル名の先頭が 00H の場合には、現在の辞書が選択されます。

拡張子 : 3文字までの長さの拡張子を大文字でセットします。

拡張子はフィールドの先頭からセットし、3文字に満たない場合には、残りにスペースをセットしてください。拡張子がない場合にはすべてスペースをセットしてください。

ES:DI には 12 バイトの領域を指すポインタをセットしてください。リターン時に、この領域には直前の辞書の情報が入ります。返される辞書の情報の形式は DS:SI にセットしたファイル名の形式と同じです。

オープンに成功すると、指定された辞書が、現在の辞書になります。失敗した場合には現在の辞書は変わりません。

辞書のクローズ

機能コード 232

機 能 辞書のクローズ

コール なし

リターン キャリーフラグがセットされた場合

AX=1 日本語入力機能がアプリケーションプログラムに開放されていません。

2 辞書がオープンされていません。

14 ディスクの読み込み中にエラーが発生しました。

キャリーフラグがセットされない場合

正しく終了しました。

解 説 現在オープン中の辞書をクローズします。

語句の登録

機能コード 233

機 能

辞書への語句登録

コール

DS:SI=読みへのポインタ (終端は NULL)

ES:DI=語句へのポインタ (終端は NULL)

リターン

キャリーフラグがセットされた場合

- AX= 1 日本語入力機能がアプリケーションプログラムに開放されていません。
- 2 辞書がオープンされていません。
- 5 読みまたは語句が長すぎます。
- 6 読みまたは語句に不正な文字が含まれています。
- 9 読みの登録されるページがありません。
- 10 登録するための領域がありません。
- 14 ディスクの読み込み中にエラーが発生しました。

キャリーフラグがセットされない場合

正しく終了しました。

解 説

オープン中の辞書に、指定された読みで、指定された語句を登録します。

DS:SI には読みへのポインタをセットしておきます。読みは 1 バイト JIS コードで、16 文字以内でなければなりません。

また、読みとして“!”, “*”, “ ” (空白), 制御コード (00H~1FH) は使用できません。読みの最後には NULL (00H) をセットしてください。部首選択部へ登録する場合は、読みの先頭に “ ” (DFH) を付けてください。

ES:DI には登録する語句へのポインタをセットしておきます。登録する語句は、シフト JIS で表わされた漢字で 16 文字以内でなければなりません。格納形式は上位バイト・下位バイトの順です (A(8260H)は、82H, 60H とセットします)。語句の最後には NULL (00H) をセットしてください。

この機能をコールした後は次候補 (機能コード 237/250)、前候補 (機能コード 238/251) は使用できません。

この語句の登録で品詞情報の登録は行えません。登録語句はすべてユーザー登録の語句になります。

語句の削除

機能コード 234

機能 辞書から語句を削除

コール DS:SI=読みへのポインタ (終端は NULL)

ES:DI=語句へのポインタ (終端は NULL)

リターン キャリーフラグがセットされた場合

AX=1 日本語入力機能がアプリケーションプログラムに開放されていません。

2 辞書がオープンされていません。

5 読みまたは語句が長すぎます。

6 読みまたは語句に不正な文字が含まれています。

7 読みまたは語句が見つかりません。

11 削除できません (システム登録の単語です)。

14 ディスク I/O エラーが発生しました。

キャリーフラグがセットされない場合

正しく終了しました。

解説 オープン中の辞書から、指定された読みの指定された語句を削除します。

DS:SI には読みへのポインタをセットしておきます。読みは 1 バイト JIS コードで 16 文字以内でなければなりません。また、読みには "!", "*", " " (空白), 制御コード (00H~1FH) は使用できません。読みの最後には NULL (00H) をセットしてください。部首選択部から削除する場合は、読みの先頭に " " (0DFH) を付けてください。

ES:DI には削除する語句へのポインタをセットしておきます。削除する語句はシフト JIS で表わされた漢字で、16 文字以内でなければなりません。格納形式は、上位バイト・下位バイトの順です (A(8260H) は、82H, 60H とセットします)。語句の最後には NULL (00H) をセットしてください。

この機能をコールした後は、次候補 (機能コード 237/250)、前候補 (機能コード 238/251) は使用できません。

この語句の削除では、システム登録の単語の削除はできません。削除語句はユーザー登録の単語に限られます。

語句の学習 (文)

機能コード 235

機 能

指定された語句の学習

コール

DS:SI =読みへのポインタ (終端は NULL)

DS:BX =読みへのポインタ (終端は NULL)

ES:DI =学習する語句へのポインタ (終端はNULL)

リターン

キャリーフラグがセットされた場合

AX=1 日本語入力機能がアプリケーションプログラムに開放されていません。

2 辞書がオープンされていません。

5 読みまたは語句が長すぎます。

6 読みまたは語句に不正な文字が含まれています。

7 読みまたは語句が見つかりません。

12 学習機能が設定されていません。

14 ディスクの読み取り中にエラーが発生しました。

キャリーフラグがセットされない場合

正しく終了しました。

解 説

オープン中の辞書内にある指定された語句を指定された読みで、候補の先頭にします (学習機能)。

DS:SI には読みへのポインタをセットしておきます。この読みは1バイト JIS コードで16文字以内でなければなりません。また、読みには"!","*"," " (空白)、制御コード (00H~1FH) は使用できません。読みの最後には NULL (00H) をセットしてください。部首選択部の語句の学習をする場合は読みの先頭に"." (0DFH) を付けてください。

DS:BX には読みへのポインタをセットしておきます。この読みはシフト JIS コードで16文字以内でなければなりません。この読みは DS:SI の指す、1バイト JIS の読みをシフト JIS に変換したものです。格納形式は、上位バイト・下位バイトの順です (A(8260H)は、82H, 60H とセットします)。1バイト JIS の読みと同様に"!","*"," " (空白) は使用できません。読みの最後には NULL (00H) をセットしてください。

部首選択部の語句の学習をする場合は読みの先頭に“ ” (814BH) を付けてください。

ES:DI には学習する語句へのポインタをセットしておきます。学習する語句はシフト JIS で表わされた漢字で 16 文字以内でなければなりません。格納形式は、上位バイト・下位バイトの順です。語句の最後には NULL (00H) をセットしてください。

この機能をコールした後は次候補 (機能コード 237)、前候補 (機能コード 238) は使用できません。必ず最初の候補 (機能コード 236) を呼び出してください。

語句の変換（最初の候補）（文）

機能コード 236

機能 指定された読みの変換（最初の候補）

コール DS:SI =読みへのポインタ（終端は NULL）
 DS:BX =読みへのポインタ（終端は NULL）
 ES:DI =変換結果格納域へのポインタ

リターン キャリーフラグがセットされた場合

- AX=1 日本語入力機能がアプリケーションプログラムに開放されていません。
 2 辞書がオープンされていません。
 5 読みが長すぎます。
 6 読みに不正な文字が含まれています。
 8 候補が見つかりません。
 14 ディスクの読み取り中にエラーが発生しました。

キャリーフラグがセットされない場合

正しく終了しました。
 AX=変換結果の長さ（バイト数）

解説 指定された読みを変換して最初の候補を返します。

DS:SI には読みへのポインタをセットしておきます。読みは1バイト JIS コードで16文字以内でなければなりません。また、読みには"!", "*", " "（空白）、制御コード(00H~1FH)は使用できません。読みの最後には NULL(00H)をセットしてください。部首選択部の変換を行う場合は、読みの先頭に "〃"(0DFH)を付けてください。

DS:BX には読みへのポインタをセットしておきます。この読みはシフト JIS コードで16文字以内でなければなりません。この読みは DS:SI の指す1バイト JIS の読みをシフト JIS に変換したものです。格納形式は、上位バイト・下位バイトの順です(A(8260H)は、82H, 60H とセットします)。1バイト JIS の読みと同様に"!", "*", " "（空白）は使用できません。カタカナは変換の対象からはずされます。読みの最後には NULL(00H)をセットしてください。部首選択部の変換を行う場合は、読みの先頭に "〃"(814BH)を付けてください。

ES:DI には変換結果の格納域へのポインタをセットしておきます。変換結果の格納形式は、上位バイト・下位バイトの順です。変換結果格納域の大きさは最大 62 バイト必要です。

読句の変換（次候補）（文）

機能コード 237

機 能 指定された読みの変換（次候補）

コール DS:SI =機能コード 236 で指定したもの
 DS:BX =機能コード 236 で指定したもの
 ES:DI =変換結果格納域へのポインタ

リターン キャリーフラグがセットされた場合

- AX=1 日本語入力機能がアプリケーションプログラムに開放されていません。
 2 辞書がオープンされていません。
 5 読みが長すぎます。
 6 読みに不正な文字が含まれています。
 8 候補が見つかりません。
 14 ディスクの読み取り中にエラーが発生しました。

キャリーフラグがセットされない場合

正しく終了しました。

AX = 変換結果の長さ（バイト数）

解 説 直前に呼び出した機能コード 236（最初の候補）237（次候補）、238（前候補）で返された候補の次の候補を返します。

DS:SI と DS:BX には機能コード 236（最初の候補）で指定したものをセットしておきます。

ES:DI には変換結果格納域へのポインタをセットしておきます。
 変換結果格納域の大きさは最大 62 バイト必要です。

読句の変換（前候補）（文）

機能コード 238

機 能

読みの変換（前候補）

コール

DS:SI = 機能コード 236 で指定したもの

DS:BX = 機能コード 236 で指定したもの

ES:DI = 変換結果格納域へのポインタ

リターン

キャリーフラグがセットされた場合

AX=1 日本語入力機能がアプリケーションプログラムに開放されていません。

2 辞書がオープンされていません。

5 読みが長すぎます。

6 読み不正な文字が含まれています。

8 候補が見つかりません。

14 ディスクの読み取り中にエラーが発生しました。

キャリーフラグがセットされない場合

正しく終了しました。

AX = 変換結果の長さ（バイト数）

解 説

直前に呼び出した機能コード 237（次候補）、238（前候補）で返された候補の前の候補を返します。

DS:SI と DS:BX には機能コード 236（最初の候補）で指定したものをセットしておきます。

ES:DI には変換結果格納域へのポインタをセットしておきます。
変換結果格納域の大きさは最大 62 バイト必要です。

日本語入力モードに入る

機能コード 239

機 能	日本語入力モードに入る
-----	-------------

コール	なし
-----	----

リターン	キャリーフラグがセットされた場合
------	------------------

AX=1 日本語入力機能がアプリケーションプログラムに開放されていません。

13 キーボードからの日本語入力が禁止されています。

キャリーフラグがセットされない場合

正しく終了しました。

解 説	日本語入力モードに入るために CTRL + XFER を押したときと同じ機能です。
-----	---

すでに日本語入力モードに入っているときは何も実行しません。

日本語入力モードから抜ける

機能コード 240

機 能 日本語入力モードから抜ける

コール なし

リターン キャリーフラグがセットされた場合

AX=1 日本語入力機能がアプリケーションプログラムに開放されていません。

キャリーフラグがセットされない場合

正しく終了しました。

解 説 日本語入力モードから抜けるために **CTRL** + **XFER** を押したときと同じ機能です。

日本語入力モードでないときは何も実行しません。

日本語入力のモードセット

機能コード 241

機能 日本語入力モードをセット

コール $AX = \boxed{b_{15}b_{14}b_{13}b_{12}b_{11}b_{10}b_9b_8b_7b_6b_5b_4b_3b_2b_1b_0}$ b_0 : 0 直接 1 間接 b_1 : 0 英数 1 カナ b_2 : 0 カタカナ 1 ひらがな b_4b_3 : 00 全角 01 半角 10 1 バイトコード b_5 : 0 1 JIS b_6 : 0 1 部首 b_7 : 0 逐次 1 連文 b_8 : 予約 $b_9 \sim b_{15}$: 未使用

リターン キャリーフラグがセットされた場合

$AX=1$ 日本語入力機能がアプリケーションプログラムに開放されていません。

キャリーフラグがセットされない場合

正しく終了しました。

 AX =実際にセットされたモード

解説 日本語入力モードに入ったときのモードセットを行います。 AX レジスタの各ビットでモードを指定します。

このとき、 b_2 は、 $b_1=1$ かつ $b_4b_3=0$ のときにのみ有効です。また $b_5=1$ のときは他のビットはすべて無効です。

b_7 は、AIかな漢字変換ドライバでのみ使用できます。

日本語入力モード取得

機能コード 242

機能 日本語入力モード取得

コール なし

リターン キャリーフラグがセットされた場合

AX=1 日本語入力機能がアプリケーションプログラムに開放されていません。

キャリーフラグがセットされない場合

正しく終了しました。

AX=現在のモード

(機能コード 241 のものに加え)

b₈: 0 1 AI 変換

解説 日本語入力モードのときの現在のモードを取得します。

モードは、リターン時の AX レジスタの各ビットで示されます。各ビットの意味は機能コード 241 (日本語入力のモードセット) を参照してください。

ビット 8 の "AI 変換" は取得のみ可能で、ビット 7 と組み合わせ、AI 逐次、AI 連文節のモードであることを示します。

2 バイト JIS をシフト JIS に変換

機能コード 243

機 能 2 バイト JIS コードをシフト JIS コードに変換

コール AX=2 バイト JIS コード

リターン キャリーフラグがセットされた場合

AX=1 日本語入力機能がアプリケーションプログラムに開放されていません。

6 語句に不正な文字が含まれています。

キャリーフラグがセットされない場合

正しく終了しました。

AX=シフト JIS コード

解 説

与えられた 2 バイト JIS コードをシフト JIS コードに変換します。
変換する JIS コードは次の範囲内でなければなりません。

- ・全角 $21\text{H} \leq \text{AH} \leq 7\text{EH}$, $21\text{H} \leq \text{AL} \leq 7\text{EH}$
- ・半角 $\text{AH} = 00\text{H}$, $21\text{H} \leq \text{AL} \leq 7\text{EH}$
または
 $\text{AH} = 00\text{H}$, $\text{A1H} \leq \text{AL} \leq \text{DFH}$

また、JIS コードの 29XXH , 2AXXH , 2BXXH も半角として扱えます (XX は $21\text{H} \sim 7\text{EH}$ の 16 進数)。

シフト JIS を 2 バイト JIS に変換

機能コード 244

機 能 シフト JIS コードを 2 バイト JIS コードに変換

コール AX=シフト JIS コード

リターン キャリーフラグがセットされた場合

AX=1 日本語入力機能がアプリケーションプログラムに開放されていません。

6 語句に不正な文字が含まれています。

キャリーフラグがセットされない場合

正しく終了しました。

AX=2 バイト JIS コード

解 説 与えられたシフト JIS コードを、2 バイト JIS コードに変換します。
変換するシフト JIS コードは次の範囲内でなければなりません。

・ $81\text{H} \leq \text{AH} \leq 9\text{FH}$, $40\text{H} \leq \text{AL} \leq \text{FCH}$ ($\text{AL} \neq 7\text{FH}$)

または

・ $\text{E0H} \leq \text{AH} \leq \text{EFH}$, $40\text{H} \leq \text{AL} \leq \text{FCH}$ ($\text{AL} \neq 7\text{FH}$)

また、シフト JIS コードの 85XXH (半角) (XX は $40\text{H} \sim \text{FCH}$ の 16 進数) は JIS コードの 29XXH , 2AXXH , 2BXXH (XX は $21\text{H} \sim 7\text{EH}$ の 16 進数) に変換されます。

逐次／連文節変換ドライバの有無取得 (逐)(連) 機能コード 247

機能	逐次／連文節変換ドライバの有無取得
----	-------------------

コール	AX=0
-----	------

リターン	キャリーフラグがセットされた場合
------	------------------

AX=1 日本語入力機能がアプリケーションプログラムに開放されていません。

キャリーフラグがセットされない場合

AX=0 AI かな漢字変換ドライバはインストールされません。

3 AI かな漢字変換ドライバが使用可能です。

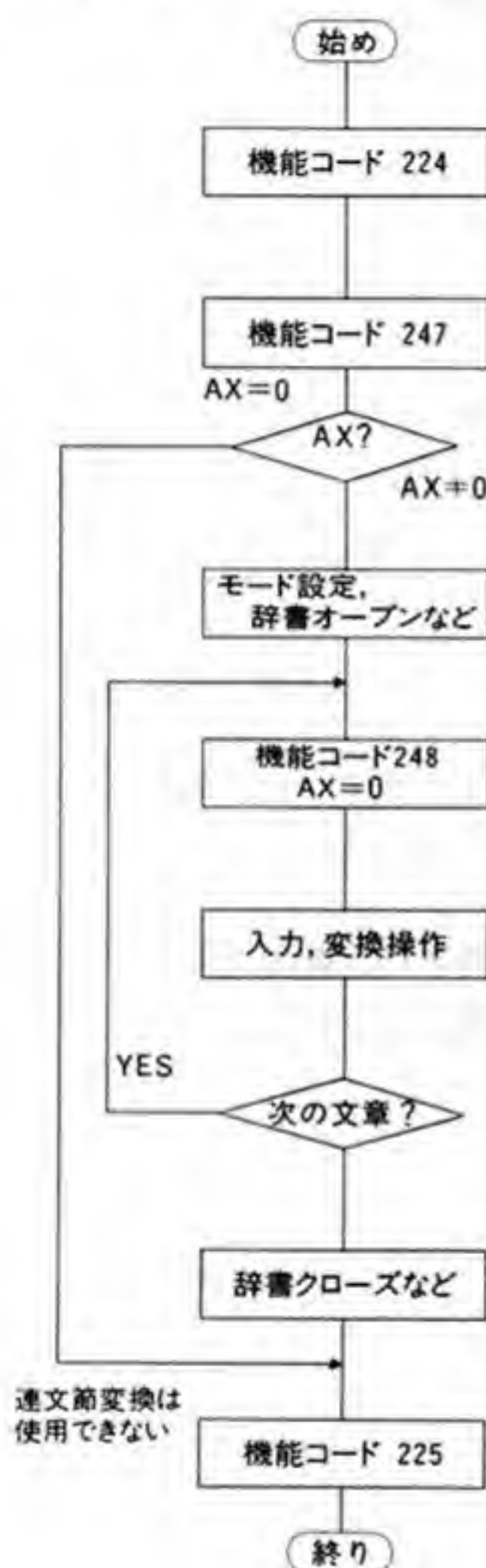
解説	AI かな漢字変換ドライバのインストールの有無を返します。
----	-------------------------------

このファンクションをコールする前に機能コード 224 (アプリケーションプログラムへの開放) をコールしておく必要があります。

このコールで AX に 3(0003H)が返された場合のみ AI かな漢字変換関連の機能 (機能コード 248-253) が使用可能となります。

注意 連文節変換機能を例として、プログラムから使用する場合に参考となる処理フローを以下に示します。

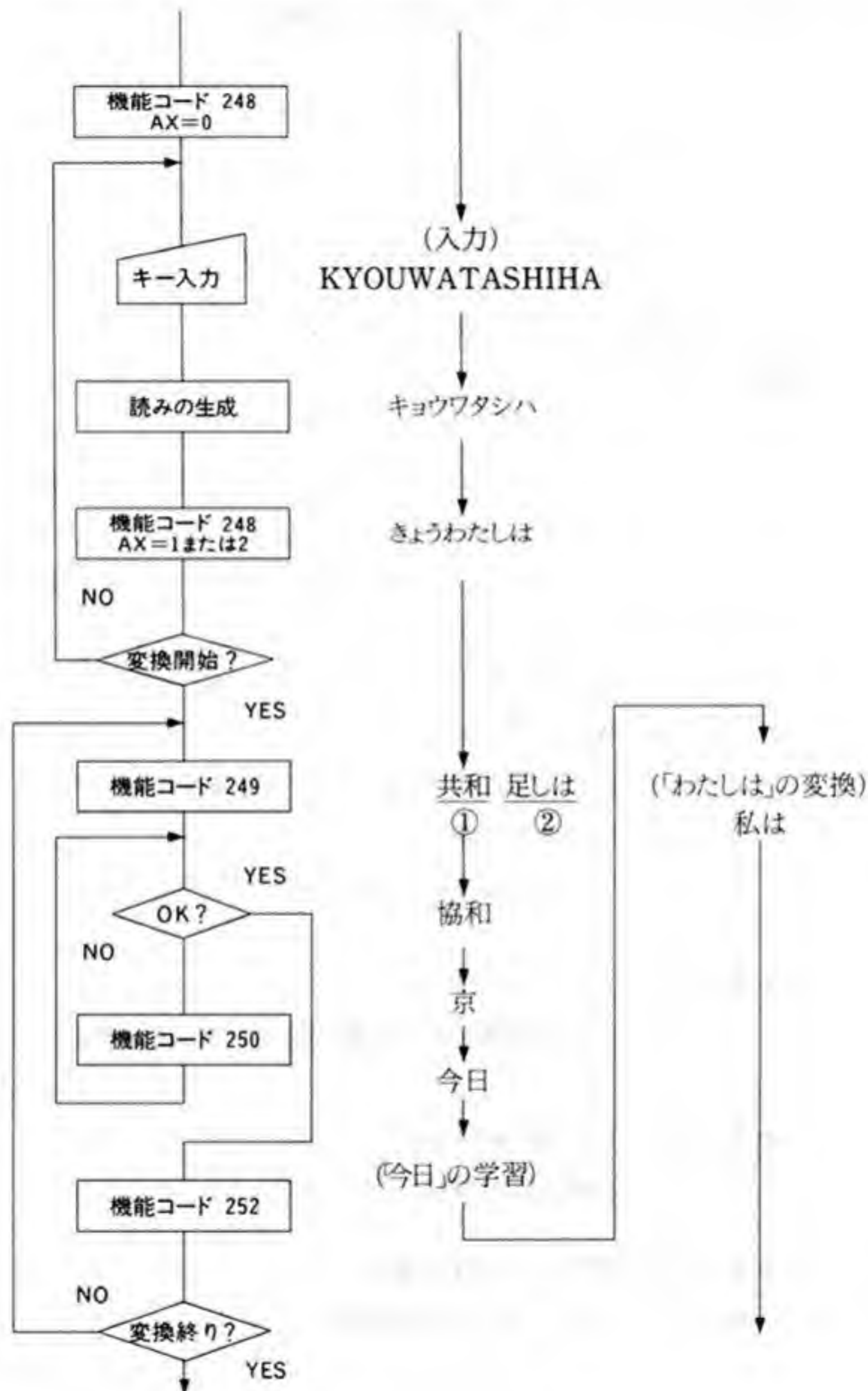
① 全体的な処理の流れ



- アプリケーションプログラムへの開放。
- AI かな漢字変換ドライバの有無取得。
- AX=0 のとき連文節変換ドライバ無し。
- 辞書オープンなどの前処理を行う。
- 先読みバッファなどの初期化。
(一連の文章の変換前に必ずコールします)
- 読みの入力, 変換, 学習など。
- 次の文章の読み入力, 変換を行う場合,
機能コード248(AX=0)から再開します。
- 辞書のクローズなどの後処理を行う。

② 変換処理の流れ

次に変換操作における処理の流れの例をフローチャートで示します。
この例では「今日私は」を変換しています。



辞書の先読み（逐）（連）

機能コード 248

機能 指定された読みによる（変換前の）辞書検索

コール AX=0 先読みバッファの初期化

- 1 辞書の先読み
- 2 読みの修正と先読み
- 3 辞書の先読みと逐次変換

DS:SI=情報テーブルへのポインタ

情報テーブル:

オフセット 00 (1 ワード)

=読みへのポインタ（終端は NULL）1 バイト読み

オフセット 02 (1 ワード)

=読みへのポインタ（終端は NULL）2 バイト読み

オフセット 04 (1 ワード)

=変換結果格納域へのポインタ（AX=3 のとき）

オフセット 06 (1 ワード)

=読みの修正箇所へのポインタ（AX=2 のとき）

オフセット 08 (1 ワード)

=すでに変換された文節を除いた 1 バイト読みへのポインタ

(AX=3 のとき)

オフセット 10 (1 ワード)

=すでに変換された文節を除いた 2 バイト読みへのポインタ

(AX=3 のとき)

オフセット 16-63, 80-127

=変換文節の情報（AX=3 のとき）

リターン キャリーフラグがセットされた場合

AX=1 逐次／連文節変換機能がアプリケーションプログラムに開放されていません。

2 辞書がオープンされていません。

5 読みが長すぎます。

6 読みに不正な文字が含まれています。

8 候補が見つかりません。

14 ディスクの読み取り中にエラーが発生しました。

(2, 5, 6, 8, 14 は、先読みバッファの初期化 (AX=0) のコールでは返されません)

キャリーフラグがセットされない場合

AX=0 正常終了しました

または、辞書の先読み機能が設定されていないので処理が行われませんでした(辞書の先読みと逐次変換のコール以外または、辞書の先読みと逐次変換のコールの場合は、逐次変換されなかった場合セットされる)。

= 逐次変換された文節の長さ (バイト数)

情報テーブル:

オフセット 14 (1 バイト)

= 逐次変換された文節数

オフセット 16-31, 80-95

= 逐次変換された文節の読み (1 バイト JIS) の長さ

オフセット 32-47, 96-111

= 逐次変換された文節の読み (シフト JIS) の長さ

オフセット 48-63, 112-127

= 逐次変換された文節の最初の候補の長さ

解 説

辞書の先読み機能が設定されている場合、読みが1文字増えるごとにこのファンクションをコールする必要があります。

このファンクションは、連文節変換ファンクション(機能コード 249)に先立ってコールされ、指定された読みのすべての読み方に対し辞書からデータを読み連文節変換ドライバ内の領域に記憶しておくためのものです。

AX=3 (辞書の先読みと逐次変換) のコールは、先読みし、記憶したデータをもとに、文節作成を行い出力可能となった文節がある場合、情報テーブルに最初の候補情報を返します。情報テーブルのオフセット 14 には、逐次変換された文節数の合計がセットされます。情報テーブルのオフセット 16-63, 80-127 には、逐次変換されたすべての各文節の長さが、セットされていきます。なお、ここでセットされた情報はそのままにしておいてください。

AX=0 (先読みバッファの初期化) のコールは、先読み機能の有無にかかわらず、その読みに対する最初の連文節変換(機能コード 249) または AX=3 のコールに先立って必ずコールしてください。情報テーブルのオフセット 00 には、読みへのポインタ (1 ワード) をセットしておきます。読みは、1 バイト JIS で 64 文字以内でなければなりません。

AX=3 と AX=1 との混在のコールはできません。

また読みには "!", "*", " " (空白), 制御コード (00H~1FH) は使用できません。読みの最後には, NULL (00H) をセットしてください。

濁音 ("が" など) や半濁音 ("ぱ" など) の 1 バイト JIS の読みは, 2 バイト ("が" なら "カ", "ぱ" なら "パ") で 1 文字として扱ってください。

情報テーブルのオフセット 02 には, 読みへのポインタ (1 ワード) をセットしておきます。この読みは, 上記 1 バイト JIS の読みをシフト JIS に変換したものでなければなりません。また, 読みの最終には NULL (00H) をセットしてください。

情報テーブルのオフセット 04 には, 変換結果の格納域へのポインタをセットしておきます。変換結果は上位バイト, 下位バイトの順にセットされます。

情報テーブルのオフセット 06 は, AX=2 のコールで参照されます。

このコールは, "AX=1 または AX=3 の" コールを行っている途中で読みの一部が変更されたときに, シフト JIS の読みの修正箇所へのポインタとして使用してください。

また, 修正箇所へのポインタは, 逐次変換された文節の読みには, 使用できません。情報テーブルのオフセット 14 の逐次変換された文節数, オフセット 16-63, 80-127 の逐次変換された読みの長さを確認し, 逐次変換されていない読みにポインタしてください。

例 1

“わたしは”を変換するときは、まず AX=0 のコールを行い連文節変換ドライバ内の先読みバッファを初期設定します。

次に AX=1 (先読み) のコールを続けます。

読みへのポインタ (1 バイト JIS)	読みへのポインタ (シフト JIS)
↓	↓
① ワ NUL	わ NUL
② ワタ NUL	わた NUL
③ ワタシ NUL	わたし NUL
④ ワタシハ NUL	わたしは NUL

“わたしは”を“わたくしは”に変更したとき、AX=2 のコールを使用します。

読みへのポインタ (1 バイト JIS)	読みへのポインタ (シフト JIS)
⑤ ワタクシハ NUL	わたくしは NUL
	↑
	修正箇所へのポインタ (“く”を指す)

* “NUL” は、NULL (00H) を表します。

例 2

辞書の先読みと逐次変換のコールにて変換された文節が出力される場合。

コール例

AX=3

情報テーブル:

オフセット	00-01	ワタシハキョウノゴ NUL
	02-03	わたしはきょうのご NUL
	04-05	

リターン例

AX=4 (4 バイト)

情報テーブル:

オフセット	04-05	私は		(変換結果(4 バイト))
	14	1 (逐次変換された文節の合計)		
	16-31	04		(第1文節 4 バイト)
	32-47	08		(第1文節 8 バイト)
	48-63	04		(第1文節 4 バイト)

続いて、読みが増え、2文節目が出力される場合。

コール例

AX=3

情報テーブル:

オフセット	00-01	ワタシハキョウノゴゴウ NUL	
	02-03	わたしはきょうのごごう NUL	
	04-05		
	08-09	キョウノゴゴウ NUL	
	10-11	きょうのごごうNUL	

リターン例

AX=6 (6 バイト)

情報テーブル:

オフセット

04-05

今日の

14

2 (逐次変換された文節数の合計)

16-31

04

04

(第1文節4バイト, 第2文節4バイト)

32-47

08

08

(第1文節8バイト, 第2文節8バイト)

48-63

04

06

(第1文節4バイト, 第2文節6バイト)

情報テーブル

オフセット

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
00																
16																
32																
48																
64																
80																
96																
112																
128																

- 00-01 読み (1 バイト JIS) へのポインタ
 02-03 読み (シフト JIS) へのポインタ
 04-05 変換結果格納域へのポインタ
 06-07 読み (シフト JIS) の修正箇所へのポインタ
 08-09 すでに変換された文節を除いた 1 バイト読みへのポインタ
 10-11 すでに変換された文節を除いた 2 バイト読みへのポインタ
 12-13 システム予約
 14 変換された文節数
 15 システム予約
 16-31 変換された各文節の読み (1 バイト JIS) のバイト数 (1~16 文節)
 32-47 変換された各文節の読み (シフト JIS) のバイト数 (//)
 48-63 変換された各文節の漢字のバイト数 (//)
 64-79 複数文節学習時の各文節のステータス (//)
 80-95 変換された各文節の読み (1 バイト JIS) のバイト数 (17~32 文節)
 96-111 変換された各文節の読み (シフト JIS) のバイト数 (//)
 112-127 変換された各文節の漢字のバイト数 (//)
 128-143 複数文節学習時の各文節のステータス (//)

連文節変換（逐）（連）

機能コード 249

機能 指定された読みの変換

コール AX = 文節番号（第1文節を0とする）

（連文節変換での最初のコールは必ず0であること。また、逐次変換ですでに変換された文節がある状態での最初のコールのときは、すでに変換されている文節数をセットしてください。）

DS : SI=情報テーブルへのポインタ

情報テーブル：

オフセット 00 (1ワード)

=読みへのポインタ（終端はNULL）

オフセット 02 (1ワード)

=読みへのポインタ（終端はNULL）

オフセット 04 (1ワード)

=変換結果格納域へのポインタ

オフセット 16-63, 80-127

=各文節の情報（AXが0以外するとき）

リターン キャリーフラグがセットされた場合

- AX= 1 逐次／連文節変換機能がアプリケーションプログラムに開放されていません。
- 2 辞書がオープンされていません。
- 5 読みが長すぎます。
- 6 読みに不正な文字が含まれています。
- 8 候補が見つかりません。または、文節数が16（または32）を超えてしまったので変換できませんでした。
- 14 ディスクの読み取り中にエラーが発生しました。

キャリーフラグがセットされない場合

正しく終了しました。

AX= 変換された文章の長さ（バイト数）

情報テーブル：

オフセット 14 (バイト数)

=文節数

オフセット 16-31, 80-95

各文節の読み (1 バイト JIS) の長さ

オフセット 32-47, 96-111

各文節の読み (シフト JIS) の長さ

オフセット 48-63, 112-127

=各文節の最初の候補の長さ

解 説

指定された読みに対して連文節変換を行ない、最初の候補を返します。連文節変換の最初のコールでは必ず AX に 0 をセットしてコールしてください。

逐次変換で、すでに変換された文節がある場合の最初のコールでは、すでに変換されている文節数を AX にセットしてください。

学習の後などに後続の文章を再変換する場合は、再変換する文章の先頭の文節番号をセットしてください。これらのとき、DS:SI で指される情報テーブルのオフセット 16-63, 80-127 には以前のコール (機能コード 249-251) のリターン情報 (各文節の読みの長さなど) をそのままセットしてください。

情報テーブルのオフセット 00 には読みへのポインタをセットしておきます。読みは 1 バイト JIS コードで 64 文字以内でなければなりません。

また読みには "!", "*", " " (空白), 制御コード (00H-1FH) は使用できません。読みの最後には NULL (00H) をセットしてください。

情報テーブルのオフセット 02 にはシフト JIS コードで 64 文字以内の読みへのポインタをセットしておきます。この読みは上記の 1 バイト JIS の読みをシフト JIS に変換したものでなければなりません。

格納形式は上位バイト, 下位バイトの順です ("A" (8260H) は 82H, 60H とセットします)。

読みには 1 バイト JIS と同様に "!", "*", " " (空白) は使用できません。また、カタカナは変換の対象からはずされています。読みの最後には必ず NULL (00H) をセットしてください。

情報テーブルのオフセット 04 には変換結果の格納域へのポインタをセットしておきます。変換結果は上位バイト, 下位バイトの順にセットされます。領域の大きさは 132 バイト確保してください。

リターン時、情報テーブルのオフセット 16 以降には変換された文節数、各文節の 1 バイト JIS の読みの長さ、シフト JIS の読みの長さ、変換された漢字の長さがそれぞれ返されます。

注意：このコールで指定した読みは、1 バイト JIS、シフト JIS とともにその読み全体の変換が終了するまで保持しておいてください。

例

・最初の連文節変換

コール例

AX=0

情報テーブル：

オフセット	00-11	→	キョウワタシハ NULL
	02-03	→	きょうわたしは NULL
	04-05	→	

リターン例

AX=10 (10 バイト)

情報テーブル：

オフセット 04-05 → 共和 足しは (変換結果(10 バイト))

14 2 (2 文節)

16-31	04	03	(第 1 文節 4 バイト, 第 2 文節 3 バイト)
32-47	08	06	(第 1 文節 8 バイト, 第 2 文節 6 バイト)
48-63	04	06	(第 1 文節 4 バイト, 第 2 文節 6 バイト)

次候補（連文節変換）（逐）（連）

機能コード 250

機能 指定された読みの変換（次候補）

コール AX=文節番号（先頭文節のとき 0）

DS:SI=情報テーブルへのポインタ

情報テーブル：

オフセット 00（1 ワード）

=機能コード 249 で指定したもの

オフセット 02（1 ワード）

=機能コード 249 で指定したもの

オフセット 04（1 ワード）

=変換結果格納域へのポインタ

オフセット 16-63, 80-127

=各文節の情報

リターン キャリーフラグがセットされた場合

AX=1 逐次／連文節変換機能がアプリケーションプログラムに開放されていません。

2 辞書がオープンされていません。

6 読みに不正な文字が含まれています。

8 候補が見つかりません。

14 ディスクの読み取り中にエラーが発生しました。

キャリーフラグがセットされない場合

正しく終了しました。

AX=次候補の長さ（バイト数）

情報テーブル：

オフセット 14（1 バイト）

=文節数

オフセット 16-31, 80-95（1 バイト×32）

=各文節の読み（1 バイト JIS）の長さ

オフセット 32-47, 96-111 (1 バイト×32)
 =各文節の読み(シフト JIS) の長さ
 オフセット 48-63, 112-127 (1 バイト×32)
 =各文節の候補の長さ

解 説

指定された番号の文節に対する次候補を返します。

AX には次候補を獲得したい文節の文節番号(機能コード 249, 250, 251 のリターン情報をもとにして) をセットします。

情報テーブルのオフセット 00-01, 02-03 には、機能コード 249 で指定したもの(読みへのポインタ) を、そのままセットしておきます。

情報テーブルのオフセット 04-05 には変換結果の格納域へのポインタをセットしておきます。結果の格納方式は機能コード 249 と同じです。

情報テーブルのオフセット 16-63, 80-127 の各文節の情報は、直前のファンクション(機能コード 249, 250, 251) で得たリターン情報を変更せずにおいてください。

該当文節の読みの長さがこのコールによって変わった場合、情報テーブル内の該当文節の読みの長さは書き換えられ、直後の文節の読みの長さを加減して余り(不足)を調整します。

例

・先頭文節の次候補

コール例 *共和(きょうわ)* → *今日(きょう)*

AX=0 (先頭文節)

情報テーブル:

オフセット 00-01 →

キョウワタシハ	NUL
---------	-----

02-03 →

きょうわたしは	NUL
---------	-----

オフセット 04-05 →

共和	足しは
----	-----

(以前の結果を入れる必要はありません)

16-31	04	03	(第1文節4バイト, 第2文節3バイト)
32-47	08	06	(第1文節8バイト, 第2文節6バイト)
48-63	04	06	(第1文節4バイト, 第2文節6バイト)

この例では、先頭文節の“きょうわ”に対する次候補を獲得しています。最初の候補は“共和”で、この文節に対して何度か次候補のコールを続けたときに得られた候補とそのときの情報テーブル内の各フィールドの値を次ページのリターン例に示してあります。

リターン例

AX=10 (10 バイト)

情報テーブル:

オフセット04-05 →

今日

16-31	03	04	(第1文節3バイト, 第2文節4バイト)
32-47	06	08	(第1文節6バイト, 第2文節8バイト)
48-63	04	06	(第1文節4バイト, 第2文節6バイト)

このリターン例では、該当文節の読みの長さが短くなっています。そのため、情報テーブルのオフセット16の内容が書き換えられ、オフセット17の内容に余り(1)が加えられています。オフセット32とオフセット33の関係も同様です。

このあと、必要ならば第1文節を学習(機能コード252)し、AX=1(第2文節目より)として連文節変換(機能コード249)をコールし、2文節目以降を再変換してください。

前候補（連文節変換）（逐）（連）

機能コード 251

機能 指定された読みの変換（前候補）

コール AX=文節番号（先頭文節のとき0）

DS:SI=情報テーブルへのポインタ

情報テーブル：

- オフセット 00（1ワード）
 =機能コード249で指定したもの
- オフセット 02（1ワード）
 =機能コード249で指定したもの
- オフセット 04（1ワード）
 =変換結果格納域へのポインタ
- オフセット 16-63, 80-127
 =各文節の情報

リターン キャリーフラグがセットされた場合

- AX= 1 逐次／連文節変換機能がアプリケーションプログラムに開放されていません。
- 2 辞書がオープンされていません。
- 6 読みに不正な文字が含まれています。
- 8 候補が見つかりません。
- 14 ディスクの読み取り中にエラーが発生しました。

キャリーフラグがセットされない場合

正しく終了しました。

AX=前候補の長さ（バイト数）

情報テーブル：

- オフセット 14（1バイト）
 =文節数
- オフセット 16-31, 80-95（1バイト×32）
 =各文節の読み（1バイトJIS）の長さ
- オフセット 32-47, 96-111（1バイト×32）
 =各文節の読み（シフトJIS）の長さ

オフセット 48-63, 112-127 (1 バイト×32)

=各文節の候補の長さ

解 説

指定された番号の文節に対する前候補を返します。

AX には前候補を獲得したい文節の文節番号(機能コード 249, 250, 251 のリターン情報をもとにして) をセットします。

情報テーブルのオフセット 00-01, 02-03 には、機能コード 249 で指定したもの(読みへのポインタ) を、そのままセットしておきます。

情報テーブルのオフセット 04-05 には変換結果の格納域へのポインタをセットしておきます。結果の格納方式は機能コード 249 と同じです。

情報テーブルのオフセット 16-63, 80-127 の各文節の情報は、直前のファンクション(機能コード 249, 250, 251) で得たリターン情報を変更せずにおいてください。

該当文節の読みの長さがこのコールによって変わった場合、情報テーブル内の該当文節の読みの長さは書き換えられ、直後の文節の読みの長さが加減され、不足(余り)が調整されます。

学習（連文節）（逐）（連）

機能コード 252

機能 指定された語句（群）の学習

コール AX = 文節番号（先頭文節のとき 0）

DX = 学習する文節数

DS : SI = 情報テーブルへのポインタ

情報テーブル：

オフセット 00 (1 ワード)

= 機能コード 249 で指定したもの

オフセット 02 (1 ワード)

= 機能コード 249 で指定したもの

オフセット 04 (1 ワード)

= 学習する語句（群）へのポインタ（終端は NULL）

オフセット 16-63, 80-127

= 各文節の情報（機能コード 249 などのリターン情報）

リターン キャリーフラグがセットされた場合

AX= 1 逐次／連文節変換機能がアプリケーションプログラムに開放されていません。

2 辞書がオープンされていません。

14 ディスクの読み取り中にエラーが発生しました。

15 論理エラーが発生しました（情報テーブルのオフセット 64-79, 128-143 を参照してください）。

情報テーブル：

オフセット 64-79, 128-143

= 学習に関する該当文節のステータス

(0 この文節は、正しく学習されました)

5 読みまたは語句が長すぎます。

6 読みまたは語句に不正な文字が含まれています。

7 読みまたは語句が見つかりません。

キャリーフラグがセットされない場合

正しく終了しました。

解 説

指定された番号の文節から指定された文節数だけ学習します。

AX には、学習する語句の文節番号を、DX には学習する文節数をセットしておきます。

情報テーブルのオフセット 00-01, 02-03 には機能コード 249 で指定した(文章全体の)読みへのポインタをセットしておきます。

情報テーブルのオフセット 04-05 には学習する語句(群)へのポインタをセットしておきます。このときの語句は、AX(文節番号)で指した文節でなければなりません。

この機能をコールした後は、次候補(機能コード 250)、前候補(機能コード 251)は使用できません。必ず辞書の先読み(機能コード 248)か連文節変換(機能コード 249)をコールしてください。

また、連続した文節の場合は、文節の区切りも学習します。

<u>今日歯医者へ</u>	文節の区切り学習
↓	
<u>今日は医者へ</u>	
 <u>興味矢崎に</u>	文節の区切り学習
↓	
<u>今日宮崎に</u>	
 <u>私寄り合いを</u>	文節の区切り学習
↓	
<u>私より愛を</u>	

例

学習の例

コール例

AX=1 (第2文節)

DX=2 (2つの文節)

情報テーブル:

オフセット	00-01	ワタシハキョウイシャニイキマス NUL			
	02-03	わたしはきょういしゃにいきます NUL			
	04-05	今日医者に NUL			
	16-31	04	03	04	04
	32-47	08	06	08	08
	48-63	--	04	06	08
	64-79				

この例では、第2文節の“今日”と第3文節の“医者に”を学習します。情報テーブルのオフセット 00-03 には連文節変換（機能コード 249）で指定したもの（読みへのポインタ）をそのままセットしてあります。オフセット 04-05 には学習する語句がセットされています。

連文節変換ドライバは、オフセット 00-03 で指される読みとオフセット 16-63 の情報から語句に対する読みを確認して学習を行います。

先読み機能の有無（連）

機能コード 253

機能 辞書の先読み機能の有無を設定

コール AX=0 先読み機能なし
 0 以外 先読み機能あり

リターン キャリーフラグがセットされた場合
 AX=1 連文節変換機能がアプリケーションプログラムに開放されていません。

キャリーフラグがセットされない場合

正しく終了しました。

AX=0 設定前は先読み機能なしでした。

1 設定前は先読み機能ありでした。

解説 連文節変換に先立つ辞書の先読み機能の有無を設定します。

辞書の先読み（機能コード 248）を使用するときには、 $AX \neq 0$ としてこの機能をコールします。先読みを行いたくない場合は $AX=0$ としてこの機能をコールしてください。

初期値は（先読み機能あり）です。

注意：AI 逐次／逐次変換のモードでは、このファンクションは無効です（常に“先読みあり”の状態となっています）。

第3章

マウスインターフェイス

3.1 イントロダクション

マウスは、パーソナルコンピュータの画面上の、カーソルの動きを容易にコントロールできるポインティングデバイスです。マウスを机上などで、自由に動かすことによって、カーソルを操作者の希望する位置へ動かすことができます。

また、マウスには、2つのボタンがついて、押したり離したりすることができます。これによって、ソフトウェア（アプリケーションプログラムなど）とのやりとりが可能になります。たとえば、カーソルを画面上のコマンドや記号の表示されている場所へ移動してボタンを押すと、ソフトウェアがそれを調べて、カーソルの位置する場所のコマンドを実行します。このように、マウスはキーボードに代わる入力装置として利用することができます。

MS-DOS では、マウスを制御するソフトウェアを、デバイスドライバで提供しています。このデバイスドライバのファイル名は "MOUSE.SYS" です。

本マニュアルでは、このマウス用デバイスドライバの機能を解説します。

3.2 マウス用デバイスドライバの組み込み

マウスを利用するには、CONFIG.SYS ファイルに次の1行を追加して、デバイスドライバを組み込みます。

```
DEVICE=MOUSE.SYS [/I:<n>]
```

[] 中の項目は省略可能。

ここで、/I:<n> は、割り込みベクタ番号の指定で、ノーマルモードでのみ有効です。<n> の値には、11H, 12H, 14H, 15H を使用することができます。ベクタ番号の指定が省略された場合、指定された番号が不適切な値であった場合には、既定値として 15H が使用されます。

なお、ハイレゾリューションモードでは、割り込みベクタ番号は 0EH に固定されています。

また、マウス用デバイスドライバは、キャラクタ系デバイスドライバですので、ADDDRV/DELDRV コマンドによって、MS-DOS の起動後に組み込み/削除することもできます (MS-DOS 3.3C ユーザーズリファレンスマニュアル参照)。

3.3 マウス用デバイスドライバについて

マウス用デバイスドライバは、マウスとアプリケーションプログラムの仲介をする働きをします。アプリケーションプログラムは、マウス用デバイスドライバのファンクションをコール（呼び出し）することによって、マウスの能力を最大限に活用することができます。

3.3.1 マウス用デバイスドライバの機能

マウス用デバイスドライバには、大きく分けて2つの機能があります。

第1は、マウスの移動距離やボタンの状態などの、マウスの情報を取得して、カーソルを移動させるなどの処理を行う機能です。

第2は、プログラムなどからの要求（ファンクションコール）によって、カーソルの位置を指定したり、ボタンの状態を通知したりする機能です。

機能には、次のような17種類があります。

機能コード	ファンクション名
10進 16進	
0 00H	環境のチェック
1 01H	カーソル表示
2 02H	カーソル消去
3 03H	カーソル位置の取得
4 04H	カーソル位置の設定
5 05H	左ボタンの押下情報の取得
6 06H	左ボタンの解放情報の取得
7 07H	右ボタンの押下情報の取得
8 08H	右ボタンの解放情報の取得
9 09H	カーソルの形の設定
11 0BH	マウスの移動距離の取得
12 0CH	ユーザー定義サブルーチンのコール条件の設定
15 0FH	ミッキー／ドット比の設定
16 10H	水平方向のカーソル移動範囲の設定
17 11H	垂直方向のカーソル移動範囲の設定
18 12H	カーソルの表示画面の設定
19 13H	グラフィック用VRANの設定と実装状況の取得

3.3.2 ファンクションコールの方法

AXレジスタに使用したいファンクションの機能コードを、その他の必要な情報を各レジスタに設定して、割り込みタイプ33H（INT33H）を実行します。

3.4 マウス用デバイスドライバのための予備知識

ここでは、マウス用デバイスドライバ（ファイル名“MOUSE.SYS”）を使用するにあたって必要となる情報を説明します。

3.4.1 接続ディスプレイの種類と解像度

マウス用デバイスドライバは、次の解像度の表示モードとディスプレイで使えます。

	横	縦
ノーマルモード専用高解像度ディスプレイ	640	400 ドット
ノーマルモードカラーグラフィックディスプレイ	640	200 ドット
ハイレゾリューションモード	1120	750 ドット

3.4.2 割り込みベクタ

ノーマルモードでは、割り込みベクタ番号を、次の1つに設定することができます。設定方法は“3.2 マウス用デバイスドライバの組み込み”を参照してください。

11H, 12H, 14H, 15H（既定値）

指定を省略した場合、指定した値が上記以外の場合は、既定値の15Hとなります。

ハイレゾリューションモードでは、割り込みベクタ番号は0EHに固定されており、変更することはできません。

3.4.3 割り込み周期

マウス用デバイスドライバでは、割り込みの周期は4種類をすべてサポートしています。ポート番号BFDBHに書き込むことにより、割り込み時間を設定することができます。

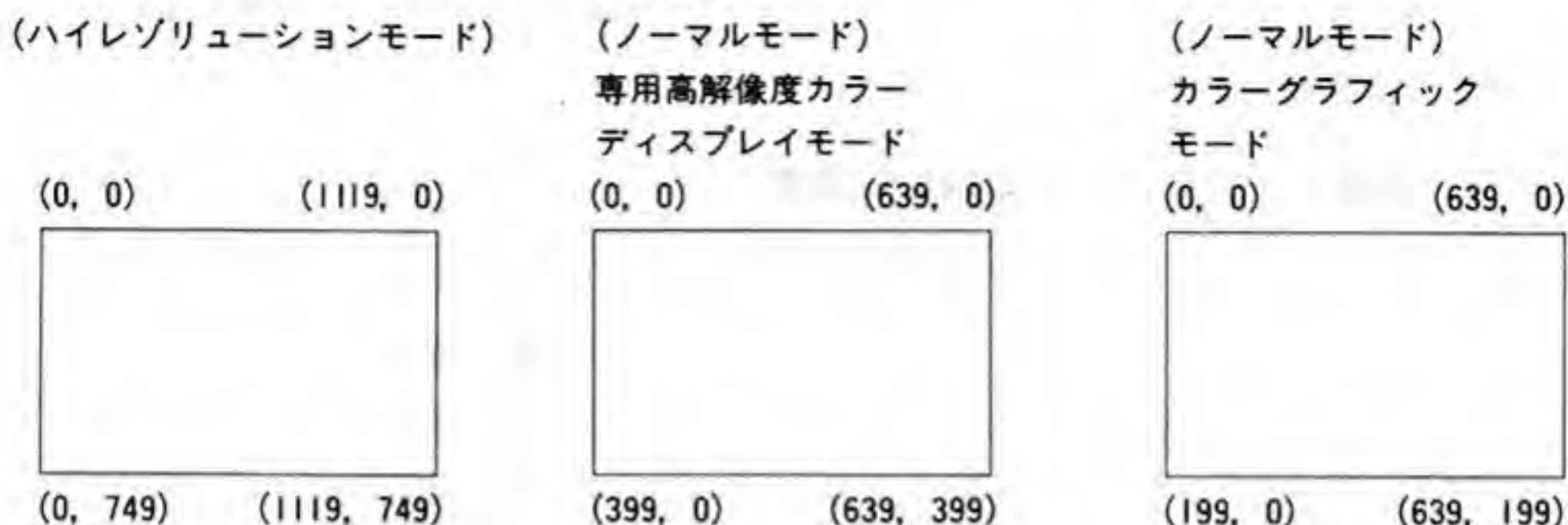
割り込み周期は、短いほどマウスの動きを正確にとらえますが、デバイスドライバのオーバーヘッドが大きくなります。

3.4.4 画面の座標系

マウス用デバイスドライバには、画面の表示モードと、接続されているディスプレイの組み合わせによって、3種類の画面モードがあります（4.1 接続ディスプレイの種類と解像度、参照）。

どの画面モードでも、スクリーン上の各点（ドット）は、1ドットを単位とした水平方向（横）と垂直方向（縦）の座標で表現します。たとえば、水平座標が5ドット目で、垂直座標が10ドット目の点の座標は（5,10）と表現します。

次に、画面モードごとに使用する座標の範囲（4隅の点の座標）を示します。



3.4.5 表示画面

このデバイスドライバでは、画面モードによって表示するグラフィック用画面（VRAM）が異なります。

ノーマルモード・高解像度ディスプレイでは、次に示すグラフィック用 VRAM（GVRAM0, GVRAM1, GVRAM2, GVRAM3）の、それぞれの開始アドレスから 32K バイトずつを、GDC 合成によって画面に表示します。

グラフィック用 VRAM の開始アドレス

GVRAM0 A8000H

GVRAM1 B0000H

GVRAM2 B8000H

GVRAM3 E0000H （オプション）

ノーマルモード・カラーグラフィックディスプレイでは、上記のグラフィック用 VRAM の、それぞれの開始アドレスから 16K バイトずつを、GDC 合成によって画面に表示します。

ハイレゾリューションモードでは、デバイスドライバは C000H から DFFF0H までの 128K バイトの画面を 4 バンク持ち、GDC 合成によって画面に表示します。

3.4.6 マウスカーソル

表示できるマウスカーソルの大きさは、画面モードによって次のように異なります。

ユーザーは、この四角形の範囲内で自由な形状に、マウスカーソルをデザインすることができます。

	横 縦
ノーマルモード・高解像度ディスプレイ	16×32 ドット
ノーマルモード・カラーグラフィックディスプレイ	16×16 ドット
ハイレゾリューションモード	16×32 ドット

マウスカーソルには、“中心点”という点があり、マウスカーソルの形状とともに設定します。これは、マウスカーソルの位置（座標）を設定／取得するための基準となる点で、上記の範囲の四角形の左上隅を(0,0)とする座標で表わします。たとえば、左上を向いた矢印では矢印の先端の点を中心点とし、十字形では縦横の線の交点を中心点に設定します。

3.4.7 ミッキー

机上でのマウスの移動距離は、“ミッキー”という単位で表わし、1 ミッキーは約 100 分の 1 インチ（約 0.25mm）です。マウス用デバイスドライバでは、マウスカーソルを画面上で 8 ドット移動させるために必要な、机上でのマウスの移動距離（ミッキー単位）を設定することで、マウスの感度を変えることができます。

3.5 マウスファンクション

ここでは、マウス用デバイスドライバの各機能ごとに、レジスタに設定すべき情報、レジスタに返される値などを解説します。

環境のチェック

機能コード 00H

機 能 マウスを使用できる環境かどうか調べる

コール AX=0

リターン AX 環境状態

0 : マウスを使用できない

-1 : マウスを使用できる

解 説 マウスが使用できる環境とは、本体にマウスインターフェイスが内蔵されている、またはマウスインターフェイスボードが接続されていて、かつマウス用デバイスドライバがメモリ中に存在している状態のことです。アプリケーションプログラムは、他のファンクションをコールする前にこのファンクションによって、環境を調べる必要があります。

またこのファンクションは、カーソルの表示、カーソルの形状、カーソルの中心点、ミッキー／ドット比、ユーザー定義サブルーチンのコール条件を初期値に設定します。

カーソル表示**機能コード 01H****機 能** マウスカーソルを画面上に表示**コール** AX=01H**リターン** なし

解 説 マウスカーソルを画面に表示させるためのファンクションです。一度このファンクションをコールすると、カーソル消去 (02H)のファンクションがコールされるまで、マウスカーソルは、マウスの動きに従って画面上を動きます。

カーソル消去

機能コード 02H

機 能	マウスカーソルを画面上から消去
-----	-----------------

コール	AX=02H
-----	--------

リターン	なし
------	----

解 説	画面上に表示されているマウスカーソルを、表示させないためのファンクションです。一度このファンクションをコールすると、カーソル表示(01H)のファンクションがコールされるまで、マウスカーソルは画面に表示されません。しかし、カーソルは表示されていない間でも、マウスを動かすことによって、画面上を移動しています。
-----	---

カーソル位置の取得

機能コード 03H

機能	現在のマウスカーソルの位置とボタンの状態を取得
コール	AX=03H
リターン	AX 左ボタンの状態 0 : 離されている -1 : 押されている BX 右ボタンの状態 0 : 離されている -1 : 押されている CX マウスカーソルの位置の水平座標 0~639 : ノーマルモードの場合 0~1119 : ハイレゾリューションモードの場合 DX マウスカーソルの位置の垂直座標 0~199 : ノーマルモード (カラーグラフィックディスプレイの場合) 0~399 : ノーマルモード (専用高解像度ディスプレイの場合) 0~749 : ハイレゾリューションモードの場合

- 解説** 現在の、マウスカーソルの位置 (座標) を得るためのファンクションです。カーソルの位置は、水平座標 (CX) と垂直座標 (DX) で得られ、値はそのときの表示モードでマウスカーソルが移動できる範囲内です。
- このファンクションでは、そのときのマウスの左右のボタンの状態 (押されているか、押されていないか) も得ることができます。

カーソル位置の設定

機能コード 04H

機 能 マウスカーソルを指定位置に移動**コール** AX=04H

CX カーソルの新しい位置の水平座標

0～639：ノーマルモードの場合

0～1119：ハイレゾリューションモードの場合

DX カーソルの新しい位置の垂直座標

0～199：ノーマルモード（カラーグラフィックディスプレイの場合）

0～399：ノーマルモード（専用高解像度ディスプレイの場合）

0～749：ハイレゾリューションモードの場合

リターン なし

解 説 マウスカーソルの位置を、指定した位置に設定するためのファンクションです。アプリケーションプログラムは、希望する位置の水平座標と垂直座標を指定して、このファンクションをコールすると、その位置にマウスカーソルが移動します。指定した位置が、マウスカーソルの移動範囲外の場合には、移動範囲内の端にカーソルを移動します。

左ボタンの押下情報の取得**機能コード 05H****機 能** マウスの左ボタンの押下情報の取得**コール** AX=05H**リターン** AX 左ボタンの状態
 0 : 放されている
 -1 : 押されている

BX 左ボタンが押された回数

CX 最後に左ボタンが押されたときのカーソル位置の水平座標

DX 最後に左ボタンが押されたときのカーソル位置の垂直座標

解 説 マウスの左ボタンの押下（押されること）状態に関する、各種の情報を取得するためのファンクションです。得られる情報は、次のとおりです。

- 左ボタンの現在の状態（押されているか、放されているか）
- このファンクションが最後にコールされてから、今回コールされるまでに、左ボタンが押された回数
- 最後に左ボタンが押されたときの、カーソルの位置（水平座標、垂直座標）

このファンクションがコールされると、マウス用デバイスドライバは、これらの情報をアプリケーションプログラムに通知します。

左ボタンの解放情報の取得

機能コード 06H

機 能 マウスの左ボタンの解放情報の取得**コール** AX=06H**リターン** AX 左ボタンの状態
 0: 放されている
 -1: 押されている

BX 左ボタンが放された回数

CX 最後に左ボタンが放されたときのカーソル位置の水平座標

DX 最後に左ボタンが放されたときのカーソル位置の垂直座標

解 説 マウスの左ボタンの解放（放されること）状態に関する、各種の情報を取得するためのファンクションです。得られる情報は、次のとおりです。

- 左ボタンの現在の状態（押されているか、放されているか）
- このファンクションが最後にコールされてから、今回コールされるまでに、左ボタンが放された回数
- 最後に左ボタンが放されたときの、カーソルの位置（水平座標、垂直座標）

このファンクションがコールされると、マウス用デバイスドライバは、これらの情報をアプリケーションプログラムに通知します。

右ボタンの押下情報の取得**機能コード 07H****機 能** マウスの右ボタンの押下情報の取得**コール** AX=07H**リターン** AX 右ボタンの状態
 0: 放されている
 -1: 押されている

BX 右ボタンが押された回数

CX 最後に右ボタンが押されたときのカーソル位置の水平座標

DX 最後に右ボタンが押されたときのカーソル位置の垂直座標

解 説 マウスの右ボタンの押下（押されること）状態に関する、各種の情報を取得するためのファンクションです。得られる情報は、次のとおりです。

- 右ボタンの現在の状態（押されているか、押されていないか）
- このファンクションが最後にコールされてから、今回コールされるまでに、右ボタンが押された回数
- 最後に右ボタンが押されたときの、カーソルの位置（水平座標、垂直座標）

このファンクションがコールされると、マウス用デバイスドライバは、これらの情報をアプリケーションプログラムに通知します。

右ボタンの解放情報の取得

機能コード 08H

機能 マウスの右ボタンの解放情報の取得**コール** AX=08H**リターン** AX 右ボタンの状態

0 : 放されている

-1 : 押されている

BX 右ボタンが放された回数

CX 最後に右ボタンが放されたときのカーソル位置の水平座標

DX 最後に右ボタンが放されたときのカーソル位置の垂直座標

解説 マウスの右ボタンの解放（放されること）状態に関する、各種の情報を取得するためのファンクションです。得られる情報は、次のとおりです。

- 右ボタンの現在の状態（押されているか、放されているか）
- このファンクションが最後にコールされてから、今回コールされるまでに、右ボタンが放された回数
- 最後に右ボタンが放されたときの、カーソルの位置（水平座標、垂直座標）

このファンクションがコールされると、マウス用デバイスドライバは、これらの情報をアプリケーションプログラムに通知します。

カーソルの形の設定

機能コード 09H

機能 マウスカーソルの形状と中心点の設定

コール AX=09H

BX カーソルの中心点の水平座標 (0~15)

CX カーソルの中心点の垂直座標

0~15 ノーマルモード (カラーグラフィックディスプレイの場合)

0~31 ノーマルモード (高解像度ディスプレイの場合)

またはハイレゾリューションモード

ES:DX カーソルの形状のデータのアドレス

リターン なし

解説 マウスカーソルの、形状と中心点を設定するためのファンクションです。カーソルの形状は、四角形のドットの集合のどのドットを表示させるかによって決まります。つまり、表示されたドットの集合が、カーソルとして見えるわけで、ユーザーはプログラムによって、四角形の範囲内で自由にカーソルの形状をデザインすることができます。たとえば、四角形のカーソルのドットのすべてのドットを表示するように指定すると、カーソルの形は四角形になります。

データの形式は、ノーマルモード (カラーグラフィックディスプレイの場合) と、ノーマルモード (高解像度ディスプレイの場合) およびハイレゾリューションディスプレイの場合で次のように異なります。

16×16 ビット ノーマルモード (カラーグラフィックディスプレイの場合)

16×32 ビット ノーマルモード (高解像度ディスプレイの場合)

16×32 ビット ハイレゾリューションモードの場合

例：カーソルの形状を決定するデータの形式

DB 11000000B, 00000000B ↑

DB 11100000B, 00000000B ノーマルモード(カラーグラフィックディスプレイの場合)は、16行、ノーマルモード(高解像度ディスプレイの場合)と、ハイレゾリューションモードは、32行

DB 00000000B, 00000000B ↓

カーソルの中心点は、カーソルの左上隅の点を原点(0,0)とした座標で指定します。この点は、カーソルの位置を検出する場合などに使用される点です。

なお、指定を省略した場合はシステムの初期設定によって、形状は左上向きの矢印、中心点は(0,0)のカーソルになります。

マウスの移動距離の取得

機能コード 0BH

機 能 ミッキー単位でのマウスの移動距離の取得

コール AX=0BH

リターン CX マウスの水平方向の移動距離
 -32768~32767 : ノーマルモード
 -1119~1119 : ハイレゾリレーションモード
 DX マウスの垂直方向の移動距離
 -32768~32767 : ノーマルモード
 -935~935 : ハイレゾリレーションモード

解 説 マウスの移動距離を取得するためのファンクションです。このファンクションが最後にコールされたときのマウスの位置から、今回コールされたときのマウスの位置までの、垂直方向と水平方向の相対的な距離をアプリケーションプログラムに通知します。

水平方向では、右の向きを正とします。垂直方向では、手前の向きを正とします。また、通知される距離の単位は、ミッキーです。

ユーザー定義サブルーチンのコール条件 の設定

機能コード 0CH

機能 ユーザー定義サブルーチンのコール条件とサブルーチンのアドレスの設定

コール AX=0CH
CX コール条件

ビット 0 カーソル位置が変化した場合

1 左ボタンが押された場合

2 左ボタンが放された場合

3 右ボタンが押された場合

4 右ボタンが放された場合

5～15 未使用

(ビット 0 を最下位ビットとして、各ビットが 1 のときにコールし、0 のときにはコールしない。)

ES:DX ユーザー定義サブルーチンのアドレス

解説 ユーザー（アプリケーションプログラム）が定義したサブルーチンを、マウス用デバイスドライバがコールする条件と、そのサブルーチンのアドレスを設定するためのファンクションです。マウス用デバイスドライバでは、次の 5 種類の現象のひとつが発生したときに、サブルーチンをコールすることができます。

- カーソル位置が変化した場合
- 左ボタンが押された場合
- 左ボタンが放された場合
- 右ボタンが押された場合
- 右ボタンが放された場合

これら 5 種類のコール条件は、同時に複数設定することができ、そのうちの 1 つが発生すると、デバイスドライバはサブルーチンをコールします。

マウス用デバイスドライバから、サブルーチンがコールされる手順は次のとおりです。

- ①マウスからの割り込みが発生すると、制御がマウス用デバイスドライバに移る。
- ②マウス用デバイスドライバは、サブルーチンのコール条件が満たされているか調べる。コール条件が満たされていなければ、次の処理へ進む。

- ③コール条件のひとつが満たされていれば、CALL FAR-PROC 命令によって、サブルーチンへ制御を移す。

・
・

(サブルーチンによる処理)

・
・

- ④サブルーチンは、RET FAR-PROC 命令によって、マウス用デバイスドライバへ制御を戻す。

マウス用デバイスドライバからサブルーチンがコールされる時、各レジスタには次のような情報が格納されています。

AX コールの原因となった条件

- 1: カーソルの位置が変化した
- 2: 左ボタンが押された
- 3: 左ボタンが放された
- 4: 右ボタンが押された
- 5: 右ボタンが放された

BL 左ボタンの状態

- 0: 放されている
- 1: 押されている

BH 右ボタンの状態

- 0: 放されている
- 1: 押されている

CX カーソル位置の水平座標

DX カーソル位置の垂直座標

注意:

マウス用デバイスドライバは、CALL FAR-PROC 命令によってサブルーチンをコールします。したがって、サブルーチンからマウス用デバイスドライバへ制御を戻すときは、RET FAR-PROC 命令を使用しなければいけません。

ミッキー／ドット比の設定

機能コード 0FH

機 能 マウスの移動距離とカーソルの移動距離の比を設定**コール** AX=0FH

CX 水平方向のミッキー／ドット比

DX 垂直方向のミッキー／ドット比

リターン なし

解 説 マウスの移動距離と、画面上のカーソルの移動距離の比を設定するファンクションです。画面上でカーソルが水平方向および垂直方向に8ドット移動するために要する、マウスの水平方向および垂直方向の移動距離を設定します。マウスの移動距離を設定する単位はミッキーで、1ミッキーは約100分の1インチ（約0.25mm）です。

ミッキー／ドット比を小さく設定すると、マウスを少し動かしただけで、カーソルは大きく移動します。大きく設定すると、マウスをかなり動かしても、カーソルは少ししか移動しません。このように、ミッキー／ドット比の設定によって、マウスの感度を変えることができます。

設定を省略した場合のシステムの初期値は、水平方向、垂直方向ともに、ミッキー／ドット比=8です。

水平方向のカーソル移動範囲の設定

機能コード 10H

機能 カーソルの水平方向の移動範囲の設定

コール AX=10H

CX 水平方向の移動範囲の最小値

0～639 : ノーマルモード

0～1119 : ハイレゾリューションモード

DX 水平方向の移動距離の最大値

0～639 : ノーマルモード

0～1119 : ハイレゾリューションモード

リターン なし

解説 画面上で、カーソルの中心点が移動できる、水平方向（左右）の範囲を設定するファンクションです。移動範囲は、最小値（左側）と、最大値（右側）で設定します。CX レジスタの値が DX レジスタの値よりも大きい場合は、DX レジスタの値が最小値、CX レジスタの値が最大値となります。

設定を省略した場合のシステムの初期設定は、画面全体です。

マウスが大きく動き、カーソルの位置が移動範囲外になったときは、移動範囲内の端にカーソルは表示されます。

垂直方向のカーソル移動範囲の設定

機能コード 11H

機能 カーソルの垂直方向の移動範囲の設定

コール AX=11H

CX 垂直方向の移動範囲の最小値

0~199: ノーマルモード (カラーグラフィックディスプレイの場合)

0~399: ノーマルモード (高解像度ディスプレイの場合)

0~749: ハイレゾリューションモード

DX 垂直方向の移動距離の最大値

0~199: ノーマルモード (カラーグラフィックディスプレイの場合)

0~399: ノーマルモード (高解像度ディスプレイの場合)

0~749: ハイレゾリューションモード

リターン なし

解説

画面上で、カーソルの中心点が移動できる、垂直方向（上下）の範囲を設定するファンクションです。移動範囲は、最小値（上側）と、最大値（下側）で設定します。CX レジスタの値が DX レジスタの値よりも大きい場合は、DX レジスタの値が最小値、CX レジスタの値が最大値となります。

設定を省略した場合のシステムの初期設定は、画面全体です。

マウスが大きく動き、カーソルの位置が移動範囲外になったときは、移動範囲内の端にカーソルは表示されます。

カーソルの表示画面の設定

機能コード 12H

機能 カーソルの表示画面（プレーン）の設定

コール AX=12H
BX 表示画面

通 常	PC-H98 で 256 色使用時
0 : プレーン 0 へ表示	0 : プレーン 0 へ表示
1 : プレーン 1 へ表示	1 : プレーン 1 へ表示
2 : プレーン 2 へ表示	2 : プレーン 2 へ表示
3 : プレーン 3 へ表示	3 : プレーン 3 へ表示
	4 : プレーン 4 へ表示
	5 : プレーン 5 へ表示
	6 : プレーン 6 へ表示
	7 : プレーン 7 へ表示

リターン なし

解説 カーソルの、表示画面を設定するためのファンクションです。カーソルの色は表示画面のパレットで設定された色になります。

PC-H98 以外の機種で、プレーン 4 以降に表示画面を指定した場合、最大プレーン（プレーン 3 または 2）に表示されます。

PC-H98 でプレーン 4 以降に表示する場合は機能コード 13H で VRAM を 8 プレーンまで使用するモードにしておく必要があります。機能コード 13H を実行せずにプレーン 4 以降への表示を指定した場合は、前回の表示画面へカーソルを表示します。

グラフィック用VRAMの設定と実装状況の取得 機能コード 13H

機能 グラフィック用 VRAM の設定と実装状況の取得

コール AX=13H

BX グラフィック VRAM の設定

- 0 : 0～2 プレーンを使用する
- 1 : 0～3 プレーンを使用する (注1)
- 2 : 0～7 プレーンを使用する (注2)

リターン **BX** グラフィック VRAM の実装状態

- 0 : 0～2 プレーンを実装
- 1 : 0～3 プレーンを実装 (注1)
- 2 : 0～7 プレーンを実装 (注2)

解説 グラフィック用 VRAM の、使用するプレーンを設定するためのファンクションです。

グラフィック用 VRAM のプレーン 3～7 が未実装の場合は、次のように動作します。

- ・プレーン 3 以降未実装時
 : プレーン 0～2 を使用
- ・プレーン 4 以降未実装でかつ、プレーン 0～7 使用要求時
 : 直前のグラフィック VRAM の設定状態となる

(注1) ハイレゾリューションモードでは、常にプレーン 0～3 を使用できます。

(注2) PC-H98 で 256 色オプションボード (PC-H98-E02) を実装している場合のみプレーン 4～7 を使用できます。

3.6 各パラメータの初期値

マウス用デバイスドライバの、種々の設定を省略した場合の設定値（初期値）は次のとおりです。

カーソル表示	：表示しない		
カーソル位置	：画面の中心		
	ノーマルモード（カラーグラフィックディスプレイの場合）	(319,99)	
	ノーマルモード（高解像度ディスプレイの場合）	(319,199)	
	ハイレゾリューションモード	(512,384)	
カーソルの形状	：左上向きの矢印		
カーソルの中心点	：(0,0)		
カーソルの表示画面	：ノーマルモード プレーン 2		
	ハイレゾリューションモード プレーン 0		
カーソルの移動範囲	：画面全体		
		水平方向	垂直方向
	ノーマルモード（カラーグラフィックディスプレイの場合）	0～639	0～199
	ノーマルモード（高解像度ディスプレイの場合）	0～639	0～399
	ハイレゾリューションモード	0～1119	0～749
ミッキー／ドット比	：8（水平方向，垂直方向ともに）		

第4章

グラフィックスドライバ

4.1 イントロダクション

MS-DOS では、PC-9800 シリーズのグラフィック機能を活用するための、基本的な描画機能を収めたグラフィックスライブラリをデバイスドライバとして提供しており、アプリケーションプログラムやユーザープログラムで利用することができます。

この章では、このグラフィックスドライバの各機能について、詳細な説明を行います。

4.2 グラフィック用デバイスドライバ

ここでは、グラフィック用デバイスドライバの組み込み方法と、利用できるファンクション（機能）の一覧を紹介します。

4.2.1 デバイスドライバの組み込み

グラフィックスドライバは、GRAPH.SYS、GRAPH.LIB の2つのファイルで構成され、デバイスドライバとして提供されます。

グラフィックスドライバを利用するには、CONFIG.SYS ファイルに次のような1行を加えてシステムを起動します。

```
DEVICE=GRAPH.SYS
```

注意：GRAPH.LIB ファイルは、カレントドライブのカレント（またはルート）ディレクトリに格納しておいてください。

また、PC-H98 用 MS-DOS では、PC-H98 上で（PC-H98 の持つ）強化されたグラフィック用ハードウェアを十分に活用するために、専用高速描画版グラフィックスドライバが提供されています。専用高速描画版グラフィックスドライバは、GRP_H98.LIB というファイル名で提供されており、このドライバを組み込むためには CONFIG.SYS のグラフィックスドライバの指定は以下のようになります。

```
DEVICE=GRAPH.SYS /F=GRP_H98.LIB[/E]
```

注意：GRP_H98.LIB ファイルは、カレントドライブのカレント（またはルート）ディレクトリに格納しておいてください。

/E は EMS が組み込まれており、かつページフレームが3ページ（48KB）以上使用可能な場合に、EMS を利用して常駐メモリを削減するためのオプションです。

PC-H98 において256色／1600万色または16色／1600万色の表示を行う場合、必ずこの専用高速描画版グラフィックスドライバを組み込む必要があります。また、256色／1600万色のカラーモードは256色オプションボード(PC-H98-E02)を実装した装置でのみ指定できます。

4.2.2 ファンクション一覧

グラフィックスドライバには、以下のファンクションが用意されています。各機能の詳細は、「4.4 ファンクションプログラミングインターフェイス」を参照してください。

●初期化ファンクション

0. グラフィックの開始
1. グラフィックの終了
2. 仮想 VRAM の生成

●環境設定ファンクション

3. 表示モードの設定
4. 描画プレーンの設定
5. 表示プレーンの設定
6. パレットの設定
7. ビューポート領域の設定
8. フォアグラウンドカラーの設定
9. バックグラウンドカラーの設定
10. ボーダカラーの設定
11. 表示スイッチの設定
12. 表示領域の設定
13. 中断処理ルーチンの設定

●描画ファンクション

14. 画面消去
15. 点の描画
16. 線の描画
17. 三角形の描画
18. 長方形の描画
19. 台形の描画
20. 円の描画
21. 楕円形の描画
22. 閉領域の塗りつぶし
23. グラフィックイメージの取得
24. グラフィックイメージの設定

25. 領域転送

26. 領域移動

●環境取得ファンクション

27. バージョンの取得

28. プレーン数の取得

29. 表示モードの取得

30. 描画プレーンの取得

31. 表示プレーンの取得

32. パレットの取得

33. ビューポート領域の取得

34. フォアグラウンドカラーの取得

35. バックグラウンドカラーの取得

36. ボーダカラーの取得

37. 表示スイッチの取得

38. 指定座標のパレットの取得

39. 表示領域の取得

40. 中断処理ルーチンの取得

4.3 ファンクションの呼び出し方法と使用例

ここでは、ユーザープログラムで、グラフィックスドライバのファンクションを利用する際の呼び出し手順と、マクロアセンブラ、C言語での使用例を紹介します。

4.3.1 ファンクションの呼び出し手順

グラフィックスドライバの各ファンクションは次の手順で呼び出します。

- (1) グラフィックスドライバのエントリテーブルの先頭アドレスを取得する。

AX レジスタ、DS:BX レジスタを次のようにセットし INT CDH を行くと、DS:BX が指す領域 (DWORD) にエントリテーブルの先頭アドレス (上位ワードにセグメント、下位ワードにオフセット) が格納されます (使用例参照)。

AX = 0

DS:BX = 任意のアドレス (セグメントとオフセット)

ただし、グラフィックスドライバが組み込まれていない場合は、動作は保障されませんので、次のデバイス名でオープン (INT 21 H, ファンクション 3 DH) することによりグラフィックスドライバの組み込みの有無を確認後、INT CDH を実行してください。

デバイス名: GRAPH ΔΔ\$

(Δ印は空白一文字を表します。)

以下に、エントリテーブルの詳細を示します。各テーブルにはそのファンクションのエントリアドレスが格納されています。

ファンクションNo	アドレス	内 容
0	0000	グラフィックの開始
1	0004	グラフィックの終了
2	0008	仮想 VRAM の生成
3	000C	表示モードの設定
4	0010	描画プレーンの設定
5	0014	表示プレーンの設定
6	0018	パレットの設定
7	001C	ビューポート領域の設定
8	0020	フォアグラウンドカラーの設定
9	0024	バックグラウンドカラーの設定
10	0028	ボーダーカラーの設定
11	002C	表示スイッチの設定
12	0030	表示領域の設定
13	0034	中断処理ルーチンの設定
14	0038	画面消去
15	003C	点の描画
16	0040	線の描画
17	0044	三角形の描画
18	0048	長方形の描画
19	004C	台形の描画
20	0050	円の描画
21	0054	楕円の描画
22	0058	閉領域の塗りつぶし
23	005C	グラフィックイメージの取得
24	0060	グラフィックイメージの設定
25	0064	領域転送
26	0068	領域移動
27	006C	バージョンの取得
28	0070	プレーン数の取得
29	0074	表示モードの取得
30	0078	描画プレーンの取得

ファンクションNo.	アドレス	内 容
31	007C	表示プレーンの取得
32	0080	パレットの取得
33	0084	ビューポート領域の取得
34	0088	フォアグラウンドカラーの取得
35	008C	バックグラウンドカラーの取得
36	0090	ボーダーカラーの取得
37	0094	表示スイッチの取得
38	0098	指定座標のパレットの取得
39	009C	表示領域の取得
40	00A0	中断処理ルーチンの取得

- (2) グラフィックスドライバで使用するデータ領域を確保する。
 下に示す形式で、データ領域（2048 バイト）を確保します。

データ領域→

パラメータブロック 48 Bytes
ワークエリア 2000 Bytes

- (3) ファンクションを呼ぶ。

(2)で確保したデータ領域内のパラメータブロックに、各ファンクションに必要なパラメータを設定します。次にデータ領域の先頭アドレスをセグメント、オフセットの順にスタックに格納して、エントリテーブル内の対応するアドレスに far call します（使用例参照）。

4.3.2 マクロアセンブラでの使用例

次に、マクロアセンブラ (MASM) によるプログラムの例を紹介します。

```

GINITIAL      EQU      00 * 4           ; グラフ開始      :
                                           ファンクション No. 0
GTERM         EQU      01 * 4           ; グラフ終了      :
                                           ファンクション No. 1
GEDSPSW       EQU      11 * 4           ; 表示スイッチ   :
                                           ファンクション No. 11
GDPSET        EQU      15 * 4           ; 点描画         :
                                           ファンクション No. 15

; データエリア

DATA          SEGMENT
GDDEV_NAME    DB        'GRAPH $'       ; グラフィックスドライバの
                                           ; デバイス名
              DB        0
GRAPH_ADDR    DD        0               ; エントリテーブルの先頭ア
                                           ; ドレス
PARA_AREA     DB        2048 DUP(0)      ; パラメータブロックとワー
                                           ; クエリア

DATA          ENDS

; スタック

STACK         SEGMENT      STACK
              DW        256 DUP(0)
STACK         ENDS

; コード部

CODE          SEGMENT
ASSUME CS : CODE, DS : DATA
START :       MOV        AX, DATA
              MOV        DS, AX
              CALL       DRV_CHK         ; ドライバの存在を確認する
              JNC        SMPL_START
              JMP        SMPL_END
SMPL_START :  CALL       GETGD_ENT        ; エントリアドレスの取得
              MOV        SI, GINITIAL    ; グラフィックスの開始
              CALL       FCALL
              MOV        BX, OFFSET DS : PARA_AREA ; 点の描画
              MOV        WORD PTR DS : [BX], 0 ; 予約パラメータ
              MOV        WORD PTR DS : 2 [BX], 0

```

```

MOV     WORD PTR DS : 4 [BX], 0      ; ラスタオペレーション番号
MOV     WORD PTR DS : 6 [BX], 1      ; 動作番号
MOV     WORD PTR DS : 8 [BX], 500    ; 点を描画する X 座標
MOV     WORD PTR DS : 0 AH [BX], 100 ; 点を描画する Y 座標
MOV     WORD PTR DS : 0 CH [BX], 7    ; パレット番号(下位 16
                                      ; BITS)
MOV     WORD PTR DS : 0 EH [BX], 0    ; パレット番号(上位 16
                                      ; BITS)

MOV     SI, GDPSET
CALL    FCALL
MOV     BX, OFFSET DS : PARA_AREA    ; 表示スイッチの設定
MOV     WORD PTR DS : 4 [BX], 1      ; 表示状態
MOV     SI, GEDSPSW
CALL    FCALL
MOV     AH, 01 H                     ; 確認のため K B 入力待ち
INT     21 H
MOV     SI, GTERM                     ; グラフィックスの終了
CALL    FCALL
SMPL_END : MOV     AX, 4 C 00 H
          INT     21 H

;      グラフィックスドライバの存在を確認する
;      (キャリーフラグが 1 ならドライバは存在しない)

DRV_CHK  PROC     NEAR
          MOV     AX, 3 D 00 H          ; ドライバをオープン
          MOV     DX, OFFSET DS : GDDEV_NAME
          INT     21 H
          JC      DRV_CHK_END          ; オープンに失敗 : 存在しない
                                      ; オープンに成功 : 存在する
          MOV     BX, AX               ; ドライバをクローズ
          MOV     AH, 3 EH
          INT     21 H
          CLC
DRV_CHK_END : RET
DRV_CHK  ENDP

;      グラフィックスドライバのエントリテーブルのアドレスを取得する
;      (GRAPH_ADDR にエントリテーブルのセグメントアドレスを格納)

GETGD_ENT PROC     NEAR
          MOV     AX, 0
          MOV     BX, OFFSET DS : GRAPH_ADDR

```



```

        INT      0 CDH
        RET
GETGD_ENT  ENDP
;          グラフィックスドライバの機能呼び出す
;          (SI にエントリテーブル上のオフセットを設定して呼び出す)
FCALL     PROC    NEAR
        MOV     BX, OFFSET DS : PARA_AREA
        PUSH    DS                ; DS 退避
        PUSH    DS                ; グラフィックスドライバへ
        PUSH    BX                ; のパラメータ
        LDS     BX, DS : GRAPH_ADDR
        CALL    DWORD PTR DS : [BX+SI]
        POP     DS                ; DS 復帰
        RET
FCALL     ENDP
CODE      ENDS
END       START

```

4.3.3 C 言語での使用例

次に、C 言語 (MS-C) によるプログラムの例を紹介します。

ただし、MS-C のバージョン 3.0 でコンパイルする場合は、オプション (/Ze) を指定する必要があります。

```

#include <stdio.h>
#include <dos.h>
#include <conio.h>
union GrpDataTag {
    unsigned int wk [1024];    /* データ領域の確保 */
    struct PsetTag {          /* 点描画のパラメータブロック */
        unsigned long Reserve;
        unsigned int Rop;
        unsigned char Act_mode;
        unsigned int X;
        unsigned int Y;
        unsigned long Color;
    } PsetTag;
    struct DspSw {             /* 表示スイッチのパラメータブロック */
        unsigned long Reserve;
        unsigned char Switch;
    } DspSw;
}

```



```

        } DspSw ;
    } GrpDataTag ;
    struct FuncEntryTag {                /* ファンクションの定義          */
        int (far pascal * FUNC [41]) (unsigned long) ;
    } far * GL ;
    FILE    * fp ;
    union    REGS    inregs, outregs ;
main () {
/*          グラフィックスドライバの存在を確認する          */
if      ((fp=fopen ("GRAPH $¥0", "r")) !=0 {
    fclose (fp) ;                      /* 存在する          */
} else {
    return (0) ;                      /* 存在しない        */
} ;

/*          エントリテーブルの先頭アドレスの取得          */
inregs. x. ax    =0 ;
inregs. x. bx    = (int) &GL ;
int 86 (0xcd, &inregs, &outregs) ;

/*          グラフィックスの開始          */
(* (GL->FUNC [0])) ((unsigned long) (int far *) &GrpDataTag) ;

/*          点の描画          */
GrpDataTag. PsetTag. Reserve    =0 ;    /* 予約パラメータ    */
GrpDataTag. PsetTag. Rop        =0 ;    /* ラスタオペレーション番号 */
GrpDataTag. PsetTag. Act_mode   =1 ;    /* 動作番号          */
GrpDataTag. PsetTag. X          =400 ;  /* 点を描画するX座標  */
GrpDataTag. PsetTag. Y          =100 ;  /* 点を描画するY座標  */
GrpDataTag. PsetTag. Color      =7 ;    /* パレット番号      */
(* (GL->FUNC [15])) ((unsigned long) (int far *) &GrpDataTag) ;
GrpDataTag. DspSw. Switch       =1 ;    /* 表示スイッチの設定 */
(* (GL->FUNC [11])) ((unsigned long) (int far *) &GrpDataTag) ;
getche () ;                      /* 確認のためのK B入力待ち */
/*          グラフィックスの終了          */
(* (GL->FUNC [1])) ((unsigned long) (int far *) &GrpDataTag) ;
} ;

```

4.4 ファンクションプログラミングインターフェイス

以下にグラフィックスドライバの各ファンクションごとに、詳細を説明します。

グラフィックの開始

ファンクション NO. 0

機 能 グラフィック専用ハードウェア、インターフェイスの初期設定をし、実行環境を整えます。

コール スタック = データ領域の先頭アドレス

リターン AX = 0 … 常に正常終了

解 説 グラフィックドライバを使用するときは、必ず本ファンクションをいちばん最初に実行してください。なお、このファンクションは画面消去は行いません。ファンクション No. 14 を必要に応じて実行してください。

具体的な初期化内容(システム VRAM および各種設定)は次のとおりです。

	ノーマルモード	ハイレゾモード
表示モード (解像度 カラーモード)	640×200 8色/8色	1120×756 16色/4096色
描画プレーン	ページ 0, プレーン 0～2	ページ 0, プレーン 0～3
表示プレーン	ページ 0, プレーン 0～2	ページ 0, プレーン 0～3
パレット	8色/8色のパレットの初期値 ^(*)	16色/4096色のパレットの初期値 ^(*)
ビューポート領域	(0, 0) - (639, 399)	(0, 0) - (1119, 935)
フォアグラウンドカラー	パレット番号 7	
バックグラウンドカラー	パレット番号 0	
ボーダーカラー	ブラック	
表示スイッチ	非表示	
表示領域	—	Y座標 0～749

(*) 詳細はパレットの設定(ファンクション NO. 6)を参照してください。

仮想 VRAM の初期化は、仮想 VRAM の生成(ファンクション No. 2)実行時に行われます。

グラフィックの終了

ファンクション NO. 1

機 能 グラフィック専用ハードウェアをリセットします。

コール スタック = データ領域の先頭アドレス

リターン AX = 0 … 常に正常終了

解 説 グラフィックスドライバの利用を終了する場合、本ファンクションを実行してください。これによりグラフィックスドライバで設定したハードウェアの状態がこの後に動作するプログラムに影響しないようにします。

仮想 VRAM の生成

ファンクション NO. 2

機 能

メモリ上に仮想的な VRAM を生成し、初期化を行います。

初期状態はノーマルモード、ハイレゾモードとも次のようになります。仮想 VRAM では解像度による区別はなく、ページは0固定となります。

表示モード モノクロ

描画プレーン ページ0、プレーン0

ビューポート領域 (0,0) - (X方向ドット数-1, Y方向ドット数-1)

フォアグラウンドカラー ... パレット番号7

バックグラウンドカラー ... パレット番号0

コール

スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	仮想 VRAM 構造体のアドレス
4H	DWORD	仮想 VRAM のアドレス
8H	WORD	X方向のドット数
0AH	WORD	Y方向のドット数
0CH	WORD	プレーン数

●仮想 VRAM 構造体のアドレス

仮想 VRAM 構造体は、仮想 VRAM のアドレスやプレーン数などの情報を格納しておく場所で、あらかじめ64バイト確保しておきます。この仮想 VRAM 構造体の先頭アドレス(上位ワード=セグメント、下位ワード=オフセット)を指定します。

●仮想 VRAM のアドレス

あらかじめ確保しておいた仮想VRAMのアドレス(上位ワード=セグメント、下位ワード=オフセット)を指定します。仮想VRAMのサイズは、次の式で求められます。このサイズは64Kを超えてもかまいません。ただし、このファンクションではメモリの管理をおこないません。サイズが64Kバイトを超える場合、連続した位置に確保しておく必要があります。

仮想 VRAM のサイズ(バイト数)

$$= ((X \text{ 方向ドット数} + 15) / 16) \times 2 \times \text{縦方向ドット数} \times \text{プレーン数}$$

● X/Y 方向のドット数

仮想 VRAM の大きさを次の範囲で指定します。

$$0 < X \text{ 方向ドット数} \leq 1120$$

$$0 < Y \text{ 方向ドット数} \leq 1120$$

● プレーン数

仮想 VRAM のプレーン数を指定します。指定したプレーン数によって、その後この仮想 VRAM に設定できるカラーモードが次のように異なります。カラーモードについては表示モードの設定(ファンクション NO.3)を参照してください。

カラーモード プレーン数	モノクロ	8 色	16 色	256 色
1	○	×	×	×
3	○ (注1)	○ (注1)	×	×
4	○	○ (注1)	○	×
8 (注2)	○	○ (注1)	○	○

8色：8色/8色および8色/4096色

16色：16色/4096色および16色/1600万色

256色：256色/1600万色

(注1) ハイレゾリューションモードでは、設定できません。

(注2) プレーン数8は、PC-H98で256色オプションボード(PC-H98-E02)を実装した装置で専用高速描画版グラフィックスドライバを利用している場合のみ指定できます。

リターン

AX = 0 … 正常終了

≠ 0 … 異常終了

指定のプレーン数が、4を越えたときなどに異常終了となります。

解説

メモリ上に仮想的な VRAM を生成し、初期化します。

仮想 VRAM と仮想 VRAM 構造体はあらかじめ確保しておいてください。

仮想 VRAM …………… 任意のサイズ

仮想 VRAM 構造体 …………… 64 バイト

仮想 VRAM に対しては、環境設定ファンクション、環境取得ファンクション、および領域転送(ファンクション NO.25)が実行できます。

表示モードの設定

ファンクション NO. 3

機 能 システム VRAM または仮想 VRAM の表示モードの設定を行います。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	対象 VRAM
4H	WORD	表示モード

	パラメータ値	表示モード	
		解像度	カラーモード
ノーマル モード	0000 H	640×200	モノクロ
	0001 H	640×200	8色／8色
	0002 H	640×200	8色／4096色
	0003 H	640×200	16色／4096色
	0100 H	640×400	モノクロ
	0101 H	640×400	8色／8色
	0102 H	640×400	8色／4096色
	0103 H	640×400	16色／4096色
	0104 H	640×400	16色／1600万色 ^(注1)
	0105 H	640×400	256色／1600万色 ^(注1)
ハイレゾ モード	0300 H	1120×750	モノクロ
	0303 H	1120×750	16色／4096色
	0304 H	1120×750	16色／1600万色 ^(注1)
	0305 H	1120×750	256色／1600万色 ^(注1)

モノクロ : ブラック、ホワイトの2色

8色／8色 : 8色のうち8色

8色／4096色 : 4096色のうち8色

16色／4096色 : 4096色のうち16色

16色／1600万色 : 1600万色のうち16色

256色／1600万色 : 1600万色のうち256色

(注1) PC-H98 で専用高速描画版グラフィックストライバを利用している場合のみ指定できます。また、256色／1600万色のカラーモードは256色オプションボード(PC-H98-E02)を実装した装置でのみ指定できます。

●対象 VRAM

システム VRAM の表示モードを設定する場合は 0，仮想 VRAM の表示モードを設定する場合は仮想 VRAM 構造体の先頭アドレス(上位ワード=セグメント，下位ワード=オフセット)を指定します。

リターン

AX = 0 … 正常終了

≠ 0 … 異常終了 (エラーコード一覧参照)

ハードウェア的に設定できないモードを指定したときは、異常終了となります。

解説

- このファンクションを実行すると、描画プレーン、表示プレーン、ビューポート、フォアグラウンドカラー/バックグラウンドカラー/ボーダーカラー、表示領域が初期化されます。これらの初期状態については、対応する環境設定ファンクションを参照してください。
- パレットの状態は、前回のそのモードでの値が引き継がれます。
- 仮想 VRAM の表示モードを設定する場合、解像度は無視されます。
- ノーマルモードでディップ SW 1-8 を OFF にして利用すると下記カラーモードの利用はできません。

8 色/4096 色

16 色/4096 色

16 色/1600万色

256 色/1600万色

描画プレーンの設定

ファンクション NO. 4

機能 システム VRAM または仮想 VRAM の実際に描画を行うページ、プレーンの設定を行います。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	対象 VRAM
4H	DWORD	描画プレーン

●描画プレーン

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	0	0	0	p7	p6	p5	p4	p3	p2	p1	p0

b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16
ページ番号								0	0	0	0	0	0	0	0

ページとはディスプレイに表示可能な画面の単位であり、プレーンとはページの構成要素です。

プレーンでは、描画したいプレーンに対応するビットを1に設定します。また、同時に複数のプレーンを指定することも可能です。ただし、指定できるプレーンはカラーモードによって異なります。

カラーモード	指定可能プレーン
モノクロ	p0～p2/p3/p7 ^(注1)
8 色	p0～p2
16 色	p0～p3
256 色 ^(注2)	p0～p7

(注1) モノクロモードの指定可能プレーンは、機種によって異なります。詳細はプレーン数の取得（ファンクション NO. 28）を参照してください。

(注2) 256 色／1600 万色のカラーモードは 256 色オプションボード (PC-H98-E02) を実装した装置で専用高速描画版グラフィックスドライバを利用している場合のみ指定できます。

ページでは、0～3までの数で指定します。ただし、仮想 VRAM に設定できるページは0のみです。システム VRAM に対して設定する場合は複数のページが利用できるため、表示するページと描画するページを別々に設定することもできます。設定できるページは解像度によって異なります。

システム VRAM に設定できるページ

解像度	ページ番号
640×200	0～3*
640×400	0～1*
1120×750	0

*設定できるページは機種によっても異なります。詳しくはプレーン数の取得(ファンクション NO.28)を参照してください。

表示モードの設定(ファンクション NO.3)を実行すると、描画プレーンは初期化されます。初期状態は、次のようにカラーモードにより異なります。

カラーモード	描画プレーン
モノクロ	ページ0, プレーン0
8色/8色, 8色/4096色	ページ0, プレーン0～2
16色/4096色, 16色/1600万色	ページ0, プレーン0～3
256色/1600万色	ページ0, プレーン0～7

●対象 VRAM

システム VRAM の表示モードを設定する場合は0、仮想 VRAM の表示モードを設定する場合は仮想 VRAM 構造体の先頭アドレス(上位ワード = セグメント, 下位ワード = オフセット)を設定します。

リターン

AX = 0 … 正常終了

≠ 0 … 異常終了(エラーコード一覧参照)

ハードウェア的に設定できないページ、プレーンを指定したときは、異常終了となります。

表示プレーンの設定

ファンクション NO. 5

機 能 システム VRAM の表示するページ、プレーンの設定を行います。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
4H	DWORD	表示プレーン

表示プレーンの指定方法は、描画プレーンと同じです。描画プレーンの設定(ファンクション NO. 4)を参照してください。

リターン AX = 0 … 正常終了
≠ 0 … 異常終了 (エラーコード一覧参照)

ハードウェア的に設定できないページ、プレーンを指定したときに、異常終了となります。

パレットの設定

ファンクション NO. 6

機 能 指定されたパレット番号に対応するカラーコードを設定します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
4H	DWORD	パレット番号
8H	DWORD	カラーコード

●パレット番号

モノクロ …… 0～1で指定します。この範囲を越えたときは、2の剰余が取られます。

8色／8色 …… 0～7で指定します。この範囲を越えたときは、8の剰余
8色／4096色 が取られます。

16色／4096色 …… 0～15で指定します。この範囲を越えた場合は16の剰余がと
16色／1600万色 られます。

256色／1600万色… 0～255で指定します。この範囲を越えた場合は256の剰余がとられます。

注意：xx色／1600万色は、PC-H98で専用高速描画版グラフィックスドライバを利用している場合のみ指定できます。また、256色/1600万色のカラーモードは256色オプションボード(PC-H98-E02)を実装した装置でのみ指定できます。

●カラーコード

b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0

red	blue
-----	------

b31 b30 b29 b28 b27 b26 b25 b24 b23 b22 b21 b20 b19 b18 b17 b16

0	0	0	0	0	0	0	0	green
---	---	---	---	---	---	---	---	-------

green, red, blue は、緑、赤、青色の各階調を表わします。

各カラーモードでの設定は、次のようになります。

モノクロ …… green, red, blue で1つでも最上位ビットが1の場合は、白色
それ以外は、黒色

8色/8色 … green, red, blue のそれぞれの最上位ビットによって、決まります。

green	red	blue	表示色
0	0	0	ブラック
0	0	1	ブルー
0	1	0	レッド
0	1	1	マゼンダ
1	0	0	グリーン
1	0	1	シアン
1	1	0	イエロー
1	1	1	ホワイト

8色/4096色, 16色/4096色 … green, red, blue の各バイトの上位4ビットだけ取り出して、それぞれ16階調とし、その組み合わせによって4096色を選択します。4ビットで表わされる0~Fは値が大きいほど明るくなります。

各モードの初期値は、次のとおりです。

モノクロ

0	0 0 0 0 0 0
1	F F F F F F

8色/8色, 8色/4096色

0	0 0 0 0 0 0
1	0 0 0 0 F F
2	0 0 F F 0 0
3	0 0 F F F F
4	F F 0 0 0 0
5	F F 0 0 F F
6	F F F F 0 0
7	F F F F F F

16色/4096色

0	0 0 0 0 0 0	8	7 7 7 7 7 7
1	0 0 0 0 F F	9	0 0 0 0 A A
2	0 0 F F 0 0	10	0 0 A A 0 0
3	0 0 F F F F	11	0 0 A A A A
4	F F 0 0 0 0	12	A A 0 0 0 0
5	F F 0 0 F F	13	A A 0 0 A A
6	F F F F 0 0	14	A A A A 0 0
7	F F F F F F	15	A A A A A A

16色/1600万色, 256色/1600万色モードの場合

green, red, blue の各階調によって 1600 万色のカラーコードを設定します。

16色/1600万色モードのパレットの初期値は16色/4096色モードのパレットの初期値と同じです。

256色/1600万色モードのパレットの初期値は16色/4096色モードのパレットの初期値を16回繰り返したものとなっています。つまりパレット0~15の色がそれぞれパレット $16 \times n \sim 16 \times n + 15$ ($n=1 \sim 15$)と同じ色となっています。

リターン

AX = 0 … 正常終了

≠ 0 … 異常終了 (エラーコード一覧参照)

解説

グラフィックドライバで色を表示するには、パレットを使用します。パレットは、それぞれのカラーモードで同時に使用できる色数だけ用意されており、パレット番号0, パレット番号1…と番号がつけられています。それぞれのパレットには、カラーコード(実際に表示される色)があらかじめ設定されており、描画ファンクションでは表示色としてこのパレット番号を指定します。

このファンクションでは、このパレット番号とカラーコードの対応を変更することができます。すでに表示されているパレット番号を変更した場合には、表示色が新しく設定されたカラーコードに変わります。

ビューポート領域の設定

ファンクション NO.7

機 能 システム VRAM または 仮想 VRAM の実際に描画される領域(ビューポート領域)を設定します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	対象 VRAM
4H	WORD	ビューポート領域左上X座標
6H	WORD	ビューポート領域左上Y座標
8H	WORD	ビューポート領域右下X座標
0AH	WORD	ビューポート領域右下Y座標

●対象 VRAM

システム VRAM のビューポート領域を設定する場合は0, 仮想 VRAM のビューポート領域を想定する場合は仮想 VRAM 構造体の先頭アドレス(上位ワード=セグメント, 下位ワード=オフセット)を指定します。

リターン AX = 0 … 正常終了
≠ 0 … 異常終了 (エラーコード一覧参照)

解 説 図形描画時に指定する座標は、ディスプレイ画面左上を原点とした、次の図のような整数系(−32768〜32767)座標ですが、実際に描画可能な領域は各解像度でディスプレイに表示できる範囲です。

ビューポート領域とは、この描画する領域のことで、このファンクションを実行することにより、この領域を変更することができます。

各解像度の最大ビューポート領域は次のとおりです。

●システム VRAM に設定する場合

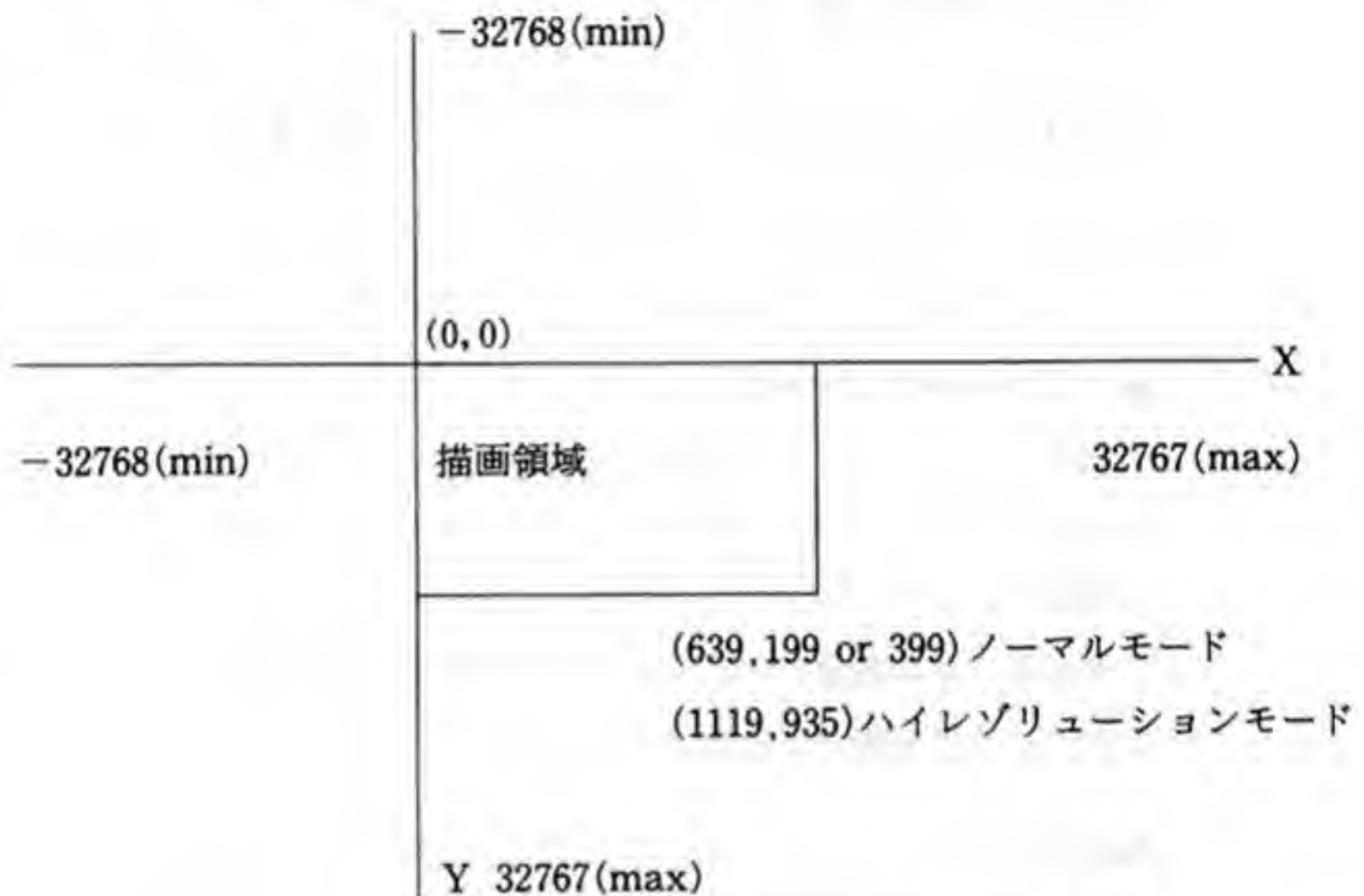
解像度	最大ビューポート領域
640×200	(0,0)-(639,199)
640×400	(0,0)-(639,399)
1120×750	(0,0)-(1119,935)

●仮想 VRAM に設定する場合

最大ビューポート領域は仮想 VRAM の生成(ファンクション NO.2)時のビューポート領域です。

このファンクションでビューポート領域を変更しても、それ以前に描画されていたグラフィックは消去されません。

また、ビューポート領域は表示モードの設定(ファンクション NO.3)を実行すると各解像度の最大ビューポート領域に初期化されます。



フォアグラウンドカラーの設定

ファンクション NO. 8

機 能 システム VRAM または 仮想 VRAM のフォアグラウンドカラーの設定を行います。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	対象 VRAM
4H	DWORD	フォアグラウンドカラー (パレット番号)

●フォアグラウンドカラー

描画ファンクションで描画色を省略したときに用いられる色で、パレット番号で指定します。パレット番号については、パレットの設定(ファンクション NO. 6)を参照してください。

表示モードの設定(ファンクション NO. 3)を実行すると、カラーモードによって次のように初期化されます。

モノクロ..... パレット番号 1

8 色 / 8 色, 8 色 / 4096 色, 16 色 / 4096 色 ... パレット番号 7

16 色 / 1600 万色, 256 色 / 1600 万色

●対象 VRAM

システム VRAM のフォアグラウンドカラーを設定する場合は 0, 仮想 VRAM のフォアグラウンドカラーを設定する場合は仮想 VRAM 構造体の先頭アドレス(上位ワード=セグメント, 下位ワード=オフセット)を指定します。

リターン AX = 0 ... 正常終了
≠ 0 ... 異常終了 (エラーコード一覧参照)

バックグラウンドカラーの設定

ファンクション NO. 9

機 能 システム VRAM または仮想 VRAM のバックグラウンドカラーの設定を行います。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	対象 VRAM
4H	DWORD	バックグラウンドカラー (パレット番号)

●バックグラウンドカラー

画面消去(ファンクション NO. 14)や領域移動(ファンクション NO. 26)で用いられる色を、パレット番号で指定します。表示モードの設定(ファンクション NO. 3)を実行すると、パレット番号 0 に初期化されます。

●対象 VRAM

システム VRAM のバックグラウンドカラーを設定する場合は 0, 仮想 VRAM のバックグラウンドカラーを設定する場合は、仮想 VRAM 構造体の先頭アドレス(上位ワード=セグメント, 下位ワード=オフセット)を指定します。

リターン AX = 0 … 正常終了
≠ 0 … 異常終了 (エラーコード一覧参照)

ボーダーカラーの設定

ファンクション NO. 10

機 能 ボーダーカラー(オーバースキャンカラー)の設定を行います。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
4H	DWORD	ボーダーカラー

●ボーダーカラー

カラーモードによって以下のように指定します。

モノクロモード

値	表示色
0	ブラック
1	ホワイト

8色／8色モード

値	表示色
0	ブラック
1	ブルー
2	レッド
3	マゼンダ
4	グリーン
5	シアン
6	イエロー
7	ホワイト

リターン AX = 0 … 正常終了
≠ 0 … 異常終了 (エラーコード一覧参照)

解 説 このファンクションは、標準ディスプレイ(解像度 640×200 ドット)接続時のみ使用できます。
表示モードの設定(ファンクション NO. 3)を実行すると、0 (ブラック)に初期化されます。

表示スイッチの設定

ファンクション NO. 11

機 能 システム VRAM のグラフィックをディスプレイに表示するか表示しないかを設定します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
4H	BYTE	表示スイッチ

●表示スイッチ

表示スイッチ = 0 非表示状態

表示スイッチ ≠ 0 表示状態

リターン AX = 0 … 正常終了
≠ 0 … 異常終了 (エラーコード一覧参照)

解 説 グラフィックの開始直後は、0 (非表示状態) となっています。

表示領域の設定

ファンクション NO. 12

機 能 システム VRAM 上の表示領域を設定します(ハイレゾリューションモードのみ)。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
4H	WORD	システム VRAM 上の Y 座標

リターン AX = 0 … 正常終了
≠ 0 … 異常終了 (エラーコード一覧参照)

解 説 この機能は、ハイレゾリューションモードでのみ使用可能です。
VRAM 上の Y 座標は、VRAM のどの位置からディスプレイに表示させるかを 0～186 の値で指定するものです。

ハイレゾリューションモードでは、VRAM のサイズが実際にディスプレイに表示されるサイズより大きいため、同時に VRAM の内容のすべてを見ることができません。この機能によって、表示領域を変えることにより、VRAM 上のどの領域でも見ることができます。



初期化時は、Y 座標 0 から 749 まで表示されます。

中断処理ルーチンの設定

ファンクション NO. 13

機 能 中断処理ルーチンのアドレスをグラフィックドライバに通知します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
4H	DWORD	中断処理ルーチンの先頭アドレス

リターン AX = 0 … 正常終了
≠ 0 … 異常終了 (エラーコード一覧参照)

解 説 各ファンクション実行中に、何らかの事象が発生したときに特別な処理を行いたい場合 ([STOP] キーが押されたらファンクションの実行を中断するなど) には、この中断処理ルーチンの設定をあらかじめ実行します。

このファンクションを実行すると、グラフィックドライバは下記ファンクション実行中に一定処理ごとに指定された中断処理ルーチンをコールします。

・閉領域の塗りつぶし (ファンクション NO. 22)

したがって、中断処理ルーチンには、中断処理を行いたい事象の発生の検出 ([STOP] キーが押されたかなど) と、事象が発生した場合に行いたい特別な処理を記述しておきます。そして、パラメータブロックにはこの中断処理ルーチンの先頭アドレス (上位ワード = セグメント, 下位ワード = オフセット) を指定します。

グラフィックの開始 (ファンクション NO. 0) 直後はグラフィックドライバ内の中断処理ルーチン (RET のみ) のアドレスが定義されています。

中断処理ルーチンの先頭アドレスは、このファンクションで再設定されるまで有効です。

中断処理ルーチンを作成するときは、次の点に注意してください。

- ① すべてのレジスタを保障してください。
- ② グラフィックドライバが管理するハードウェア (グラフィック VRAM, GDC, グラフィックチャージャ, EGC) の状態を変更しないでください。

- ③ このルーチンの中で、グラフィックスドライバのファンクションをコールしないでください。
- ④ グラフィックスドライバのデータ領域を変更しないでください。
- ⑤ 中断処理ルーチンからリターンしない場合は、ユーザー側でスタックを解決してください。

画面消去

ファンクション NO. 14

機 能 システム VRAM に描画されている内容を消去します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	予約パラメータ(必ず0を指定してください)

リターン AX = 0 … 正常終了
≠ 0 … 異常終了 (エラーコード一覧参照)

解 説 消去されるのは、ビューポート領域のみです。このとき、バックグラウンドカラーが使用されます。バックグラウンドカラーの設定(ファンクション NO. 9)を参照してください。

点の描画

ファンクション NO. 15

機 能 システム VRAM 上の指定の座標に指定のパレット番号で点を描画します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	予約パラメータ (必ず 0 を指定してください)
4H	WORD	ラストオペレーション番号
6H	BYTE	動作番号
8H	WORD	点を描画する X 座標
0AH	WORD	点を描画する Y 座標
0CH	DWORD	点の色を表わすパレット番号

●ラストオペレーション(ROP)番号

次の 16 種類の中から指定できます。

ROP 番号	論理演算
0000	S
0001	\bar{S}
0002	D
0003	\bar{D}
0004	$D + S$
0005	$D + \bar{S}$
0006	$\bar{D} + S$
0007	$\bar{D} + \bar{S}$
0008	$D \cdot S$
0009	$D \cdot \bar{S}$
000A	$\bar{D} \cdot S$
000B	$\bar{D} \cdot \bar{S}$
000C	$D (+) S$
000D	$D (+) \bar{S}$
000E	$\bar{D} (+) S$
000F	$\bar{D} (+) \bar{S}$

D : VRAM 上の現在のパレット番号

\bar{D} : D を反転(NOT)したパレット番号

S : 描画するパレット番号

\bar{S} : S を反転(NOT)したパレット番号

+ : OR(論理和)

• : AND(論理積)

(+) : XOR(排他的論理和)

●動作番号

01H=点の色を省略したとき、フォアグラウンドカラーを使用します。

02H=点の色を省略したとき、バックグラウンドカラーを使用します。

●X座標/Y座標

描画したい点を整数系(−32768〜32767)座標で指定します。

●点の色

表示したい点の色をパレット番号で指定します。点の色を省略したいときは、最上位ビットに1を設定してください。

リターン

AX = 0 … 正常終了

≠ 0 … 異常終了 (エラーコード一覧参照)

動作番号に上記以外の値が設定されたときは、異常終了となります。

解説

ラストオペレーションは色を表わすパレットに対して働くもので、全ての描画ファンクションで使用できます。

例えば、カラーモードが16色/4096色で座標(100,100)のパレット番号が2であるときに、この座標にパレット番号1で点を描画すると、ラストオペレーション番号によって以下のように描画後のパレット番号が異なります。

ラストオペレーション番号	描画後のパレット番号
0	1
1	14
2	2
3	13
4	3
5	14
6	13
7	15
8	0
9	2
10	1
11	12
12	3
13	12
14	12
15	3

パレット番号1の反転(NOT)は、モノクロの場合パレット番号0となり、8/8色、8/4096色の場合は6となります。

線の描画

ファンクション NO. 16

機能 システム VRAM 上の指定した 2 点を結ぶ直線を、指定されたパレット番号、線種パターン、線幅で描画します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	予約パラメータ (必ず 0 を指定してください)
4H	WORD	ラストオペレーション番号
6H	WORD	描画始点 X 座標
8H	WORD	描画始点 Y 座標
0AH	WORD	描画終点 X 座標
0CH	WORD	描画終点 Y 座標
0EH	BYTE	描画フラグ
10H	DWORD	線の色を表わすパレット番号
14H	WORD	線幅
16H	BYTE	線種パターン長
18H	WORD	線種パターン
1AH	WORD	拡張線種パターン

●描画フラグ

00H = 線種パターン、線幅省略(幅 1 ドットの実線)

01H = 線種パターンのみ指定(幅 1 ドット)

10H = 線幅のみ指定(実線)

11H = 線種パターン、線幅指定

●線幅

0～15 の値で指定します(線幅 + 1)ドットの幅で線が描画されます。

線幅は、幅 1 ドットの線を中心として、描画する線が仰角 45 度以下の場合は上下方向に、45 度以上の場合は水平方向に描画されます。よって指定した線幅が奇数の場合には、上下方向に描画される場合は下側、水平方向に描画される場合は右側が、その反対側より 1 ドット分多くなります。

●線種パターン長

10H = 16 ビット(線種パターンを使用する)

20H = 32 ビット(線種パターンと拡張線種パターンを使用する)

●線種パターン、拡張線種パターン

破線などの線種をビットパターンで指定します。直線は指定されたビットパターンを繰り返しドットに対応させて描画されます。このビットパターンは線種パターン長で 16 ビット単位か 32 ビット単位に指定できます。拡張線種パターンは 32 ビット単位のとくに指定します。

●ラストオペレーション番号

点の描画(ファンクション NO.15)を参照してください。

●X 座標/Y 座標

描画したい線の始点と終点を整数系(−32768~32767)座標で指定します。

●線の色

描画したい線の色をパレット番号で指定します。

最上位ビットに 1 を設定するとフォアグラウンドカラーで描画されます。

リターン

AX = 0 … 正常終了

≠ 0 … 異常終了(エラーコード一覧参照)

解説

指定した 2 点を結ぶ直線を、指定されたパレット番号、線種パターン、線幅で描画します。線種パターンとディスプレイ上のドットイメージの関係は次の例のようになります。

16 ビット単位で線種パターンを使用する場合

線種パターン長: 10H

線種パターン : 0F0EH

ディスプレイ上のドットイメージ

○○○○●●●●○○○○●●●○

32 ビット単位で線種パターンを使用する場合

線種パターン長 : 20H

線種パターン : 0F0EH

拡張線種パターン: 0C08H

ディスプレイ上のドットイメージ

○○○○●●●●○○○○●●●○○○○○○●●○○○○○○●○○○

三角形の描画

ファンクション NO. 17

機 能 システム VRAM 上の指定した3点を結ぶ三角形を描画します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	予約パラメータ(必ず0を指定してください)
4H	WORD	ラストオペレーション番号
6H	WORD	第1 X座標
8H	WORD	第1 Y座標
0AH	WORD	第2 X座標
0CH	WORD	第2 Y座標
0EH	WORD	第3 X座標
10H	WORD	第3 Y座標
12H	BYTE	描画フラグ
14H	DWORD	線の色を表わすパレット番号
18H	WORD	線幅
1AH	BYTE	線種パターン長
1CH	WORD	線種パターン
1EH	WORD	拡張線種パターン
20H	BYTE	塗りつぶしフラグ
22H	DWORD	塗りつぶす色を表わすパレット番号
22H	WORD	塗りつぶしに使用するタイルパターンの長さ
24H	DWORD	タイルパターンの格納域のアドレス

●塗りつぶしフラグ

00H = 塗りつぶしません。

01H = 指定のパレット番号で塗りつぶします(省略時は、線の色と同じ色で塗りつぶします。)

02H = 指定のタイルパターンで1バイト単位に塗りつぶします。

03H = 指定のタイルパターンで2バイト単位に塗りつぶします。

●タイルパターン長

タイルパターン長は、モノクロのときは1以上、8色/8色、8色/4096色のときは3以上、16色/4096色のときは4以上の長さをバイト数で指定してください。

●タイルパターン格納域

プレーン0から順番に塗りつぶすタイルパターンを、1バイト単位(あるいは2バイト単位)で格納します。

パラメータブロックのオフセット24Hのダブルワードには、このタイルパターン格納域のアドレス(上位ワード=セグメント、下位ワード=オフセット)を格納します。

●ラストオペレーション番号

点の描画(ファンクション NO. 15)を参照してください。

●X座標/Y座標

描画したい三角形の各頂点を整数系(-32768~32767)座標で指定します。

●描画フラグ、線の色、線幅、線種パターン長、線種パターン、拡張線種パターン

線の描画(ファンクション NO. 16)を参照してください。

リターン

AX = 0 ... 正常終了

≠ 0 ... 異常終了(エラーコード一覧参照)

解説

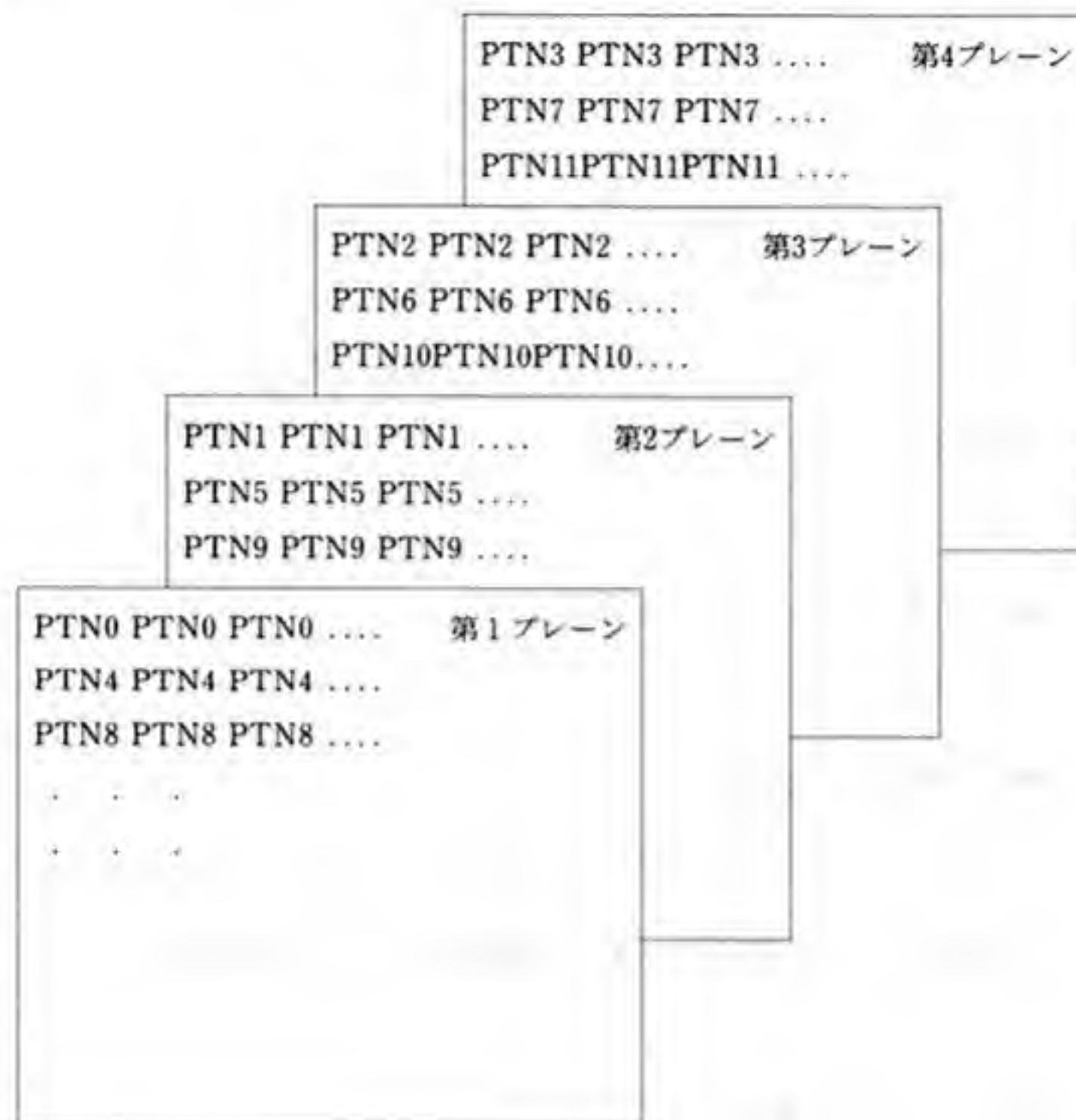
指定された3点を頂点とした三角形を描き、必要であれば、内部を塗りつぶします。

タイルパターンは、プレーン0から順番に各プレーンに対応しています。

たとえば、n+1バイトのタイルパターンが次のように格納されているとします。

PTN0	PTN1	PTN2	...	PTNn
------	------	------	-----	------

これを、16色/4096色で画面上に1バイト単位で展開させると、次のようになります。



塗りつぶしフラグの設定によって2バイト単位に展開させることもできます。
 タイルパターンの長さが、条件より小さいときは処理は行いません。
 タイルパターンの長さと各表示モードのプレーン数が対応しない場合、余りは、
 無視されます。

ラスタオペレーションで0以外の値を指定すると、頂点が正確に描画できない
 ことがあります。

長方形の描画

ファンクション NO. 18

機 能 システム VRAM 上の指定された2点を結ぶ線分を対角線とする長方形を描画します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	予約パラメータ(必ず0を指定してください)
4H	WORD	ラストオペレーション番号
6H	WORD	線分の開始X座標
8H	WORD	線分の開始Y座標
0AH	WORD	線分の終了X座標
0CH	WORD	線分の終了Y座標
0EH	BYTE	描画フラグ
10H	DWORD	線の色を表わすパレット番号
14H	WORD	線幅
16H	BYTE	線種パターン長
18H	WORD	線種パターン
1AH	WORD	拡張線種パターン
1CH	BYTE	塗りつぶしフラグ
1EH	DWORD	塗りつぶす色を表わすパレット番号
1EH	WORD	塗りつぶしに使用するタイルパターンの長さ
20H	DWORD	タイルパターンの格納域のアドレス

●ラストオペレーション番号

点の描画(ファンクション NO.15)を参照してください。

●X座標/Y座標

描画したい長方形の対角線の始点/終点を整数系(-32768~32767)座標で指定します。

●描画フラグ, 線の色, 線幅, 線種パターン長, 線種パターン, 拡張線種パターン

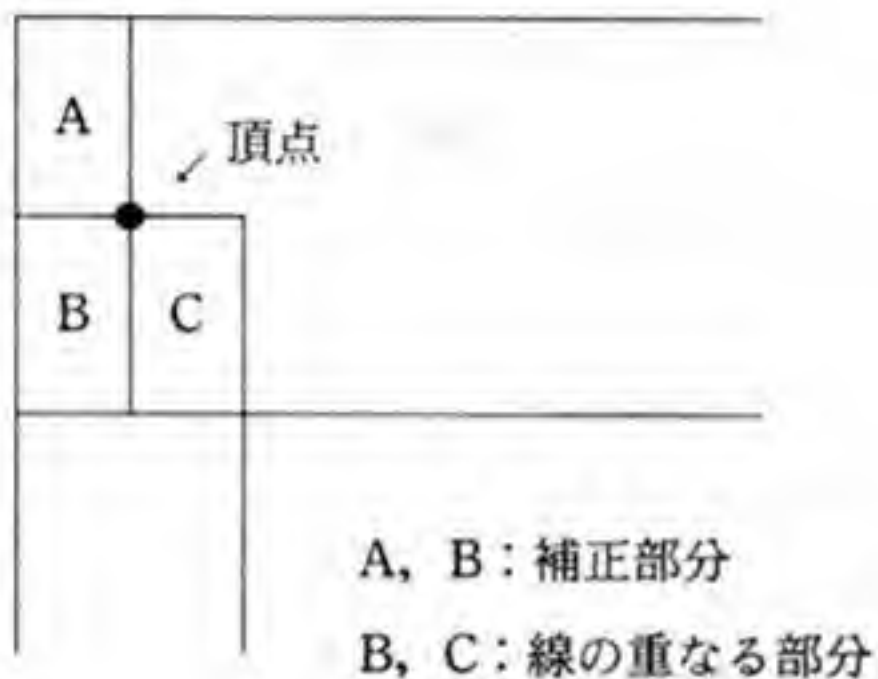
線の描画(ファンクション NO.16)を参照してください。

- 塗りつぶしフラグ, 塗りつぶす色, タイルパターンの長さ, タイルパターン格納域
三角形の描画(ファンクション NO.17)を参照してください。

リターン $AX = 0 \dots$ 正常終了
 $\neq 0 \dots$ 異常終了(エラーコード一覧参照)

解 説 指定された2点を結ぶ線分を対角線とする長方形を描画し、必要であれば、内部を塗りつぶします。

線幅を指定する場合は、各頂点で水平方向の線による補正を行なうため、頂点は次図のようになります。



ラストオペレーションで0以外の値を指定すると、頂点が正確に描画されないことがあります。

台形の描画

ファンクション NO. 19

機能 システム VRAM 上の指定された 4 点を頂点とする台形を描画します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	予約パラメータ(必ず 0 を指定してください)
4H	WORD	ラストオペレーション番号
6H	WORD	第 1 X座標
8H	WORD	第 1 Y座標
0AH	WORD	第 2 X座標
0CH	WORD	第 3 X座標
0EH	WORD	第 3 Y座標
10H	WORD	第 4 X座標
12H	BYTE	描画フラグ
14H	DWORD	線の色を表わすパレット番号
18H	WORD	線幅
1AH	BYTE	線種パターン長
1CH	WORD	線種パターン
1EH	WORD	拡張線種パターン
20H	BYTE	塗りつぶしフラグ
22H	DWORD	塗りつぶす色を表わすパレット番号
22H	WORD	塗りつぶしに使用するタイルパターンの長さ
24H	DWORD	タイルパターンの格納域のアドレス

リターン AX = 0 … 正常終了
≠ 0 … 異常終了 (エラーコード一覧参照)

解説 指定された 4 点を頂点とすると台形を描画します。必要があれば、内部を塗りつぶします。

各パラメータについては点の描画(ファンクション NO.15)、線の描画(ファンクション NO.16)、三角形の描画(ファンクション NO.17)などを参照してください。

ラスタオペレーションで0以外の値を指定すると、頂点が正確に描画されないことがあります。

円の描画

ファンクション NO. 20

機能 システム VRAM 上の指定された中心点と半径をもとに、円、円弧、扇形を描画します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	予約パラメータ (必ず 0 を指定してください)
4H	WORD	ラストオペレーション番号
6H	WORD	中心点 X 座標
8H	WORD	中心点 Y 座標
0AH	WORD	半径
0CH	WORD	開始点 X 座標
0EH	WORD	開始点 Y 座標
10H	WORD	終了点 X 座標
12H	WORD	終了点 Y 座標
14H	BYTE	描画フラグ
16H	DWORD	線の色を表わすパレット番号
1AH	BYTE	形状フラグ
1CH	BYTE	線種パターン長
1EH	WORD	線種パターン
20H	WORD	拡張線種パターン
22H	BYTE	塗りつぶしフラグ
24H	DWORD	塗りつぶす色を表わすパレット番号
24H	WORD	塗りつぶしに使用するタイルパターンの長さ
26H	DWORD	タイルパターンの格納域のアドレス

●形状フラグ

b7 b6 b5 b4 b3 b2 b1 b0

0	0	0	×	×	×	×	×
---	---	---	---	---	---	---	---

- b0=開始座標フラグ 0… 開始座標指定なし
1… 開始座標指定あり
- b1=開始線分フラグ 0… 開始点と中心点に線分を描画しない
1… 開始点と中心点間に線分を描画する
- b2=終了座標フラグ 0… 終了座標指定なし
1… 終了座標指定あり
- b3=終了線分フラグ 0… 終了点と中心点間に線分を描画しない
1… 終了点と中心点間に線分を描画する
- b4=描画範囲フラグ 0… 開始点, 終了点が等しい場合は, 全円を描画する
1… 開始点, 終了点が等しい場合は, 開始点(終了点)を描画する

●ラストオペレーション番号

点の描画(ファンクション NO.15)を参照してください。

●中心点, 半径, 開始点, 終了点

描画したい円の中心点と半径を整数系(-32768~32767)座標で指定します。円弧, 扇型を描画したい場合はさらに開始点と終了点も指定します。

●描画フラグ

00H = 線種パターン省略(実線)

01H = 線種パターン指定

●線の色, 線種パターン長, 線種パターン, 拡張線種パターン

線の描画(ファンクション NO.16)を参照してください。

●塗りつぶしフラグ, 塗りつぶす色, タイルパターンの長さ, タイルパターン格納域

三角形の描画(ファンクション NO.17)を参照してください。

リターン

AX = 0… 正常終了

≠ 0… 異常終了(エラーコード一覧参照)

解説

指定された中心点, 半径をもとに円を描画します。また開始点, 終了点を指定することにより円弧, 扇形の描画も可能です。

ラストオペレーションで0以外の値を指定すると, 頂点が正確に描画されないことがあります。

楕円の描画

ファンクション NO. 21

機 能 システム VRAM 上の指定された中心点と X 方向半径および Y 方向半径をもとに、楕円、楕円弧を描画します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	予約パラメータ (必ず 0 を指定してください)
4H	WORD	ラストオペレーション番号
6H	WORD	中心点 X 座標
8H	WORD	中心点 Y 座標
0AH	WORD	X 方向半径
0CH	WORD	Y 方向半径
0EH	WORD	開始点 X 座標
10H	WORD	開始点 Y 座標
12H	WORD	終了点 X 座標
14H	WORD	終了点 Y 座標
16H	BYTE	描画フラグ
18H	DWORD	線の色を表わすパレット番号
1CH	BYTE	形状フラグ
1EH	BYTE	線種パターン長
20H	WORD	線種パターン
22H	WORD	拡張線種パターン
24H	BYTE	塗りつぶしフラグ
26H	DWORD	塗りつぶす色を表わすパレット番号
26H	WORD	塗りつぶしに使用するタイルパターンの長さ
28H	DWORD	タイルパターンの格納域のアドレス

リターン AX = 0 … 正常終了
≠ 0 … 異常終了 (エラーコード一覧参照)

解説 指定された中心点とX方向半径およびY方向半径をもとに、楕円を描画します。
また、開始点／終了点を指定することにより楕円弧の描画もできます。

半径をX方向、Y方向の2つ指定すること以外は円の描画と同じです。円の描画(ファンクション NO.20)を参照してください。

閉領域の塗りつぶし

ファンクション NO. 22

機 能 システム VRAM 上の指定された座標を含み、指定された境界色で囲まれる閉領域を塗りつぶします。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	予約パラメータ(必ず 0 を指定してください)
4H	WORD	ラストオペレーション番号
6H	WORD	描画開始 X 座標
8H	WORD	描画開始 Y 座標
0AH	DWORD	境界色を表わすパレット番号
0EH	BYTE	塗りつぶしフラグ
10H	DWORD	塗りつぶす色を表わすパレット番号
10H	WORD	塗りつぶしに使用するタイルパターンの長さ
12H	DWORD	タイルパターンの格納域のアドレス
16H	DWORD	作業域先頭アドレス
1AH	WORD	作業域の大きさ(バイト数)

●描画開始座標

塗りつぶしたい閉領域内の任意の点を整数系(−32768〜32767)座標で指定します。

●境界色

塗りつぶしたい領域の境界線の色をパレット番号で指定します。最上位ビットに 1 を設定するとフォアグラウンドカラーを採用します。

●作業域

作業域はあらかじめ確保しておき、そのアドレス(上位ワード=セグメント、下位ワード=オフセット)と、その大きさをバイト数で指定します。

その他のパラメータについては点の描画(ファンクション NO. 15)、三角形の描画(ファンクション NO. 17)を参照してください。

リターン $AX = 0 \dots$ 正常終了
 $\neq 0 \dots$ 異常終了 (エラーコード一覧参照)

解 説 作業域の大きさは、16 バイト以上必要で、塗りつぶす閉領域の形状によって異なります。作業域が不足した場合は、エラーコード 2 を返し処理を中断します。その場合は、作業域を十分大きくとるか、分割して塗りつぶしてください。
 ビューポート領域外に開始座標を指定した場合は、何も描画されません。

グラフィックイメージの取得

ファンクション NO. 23

機 能 システム VRAM 上の指定された 2 点結ぶ線分を対角線とする矩形領域のイメージを、指定されたメモリ上の格納域に格納します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	予約パラメータ (必ず 0 を指定してください)
4H	WORD	左上 X 座標
6H	WORD	左上 Y 座標
8H	WORD	右下 X 座標
0AH	WORD	右下 Y 座標
0CH	WORD	メモリ上の格納域の長さ
0EH	DWORD	メモリ上の格納域のアドレス

●左上/右下座標

グラフィックイメージを取得したい矩形の左上と右下の頂点を整数系 (−32768〜32767) 座標で指定します。

●格納域の長さ

メモリ上にあらかじめ確保しておいた格納域の長さをバイト数で指定します。ただし、指定された矩形領域が

$$(x1, y1) - (x2, y2)$$

とすると、次の条件を満たしていなければなりません。

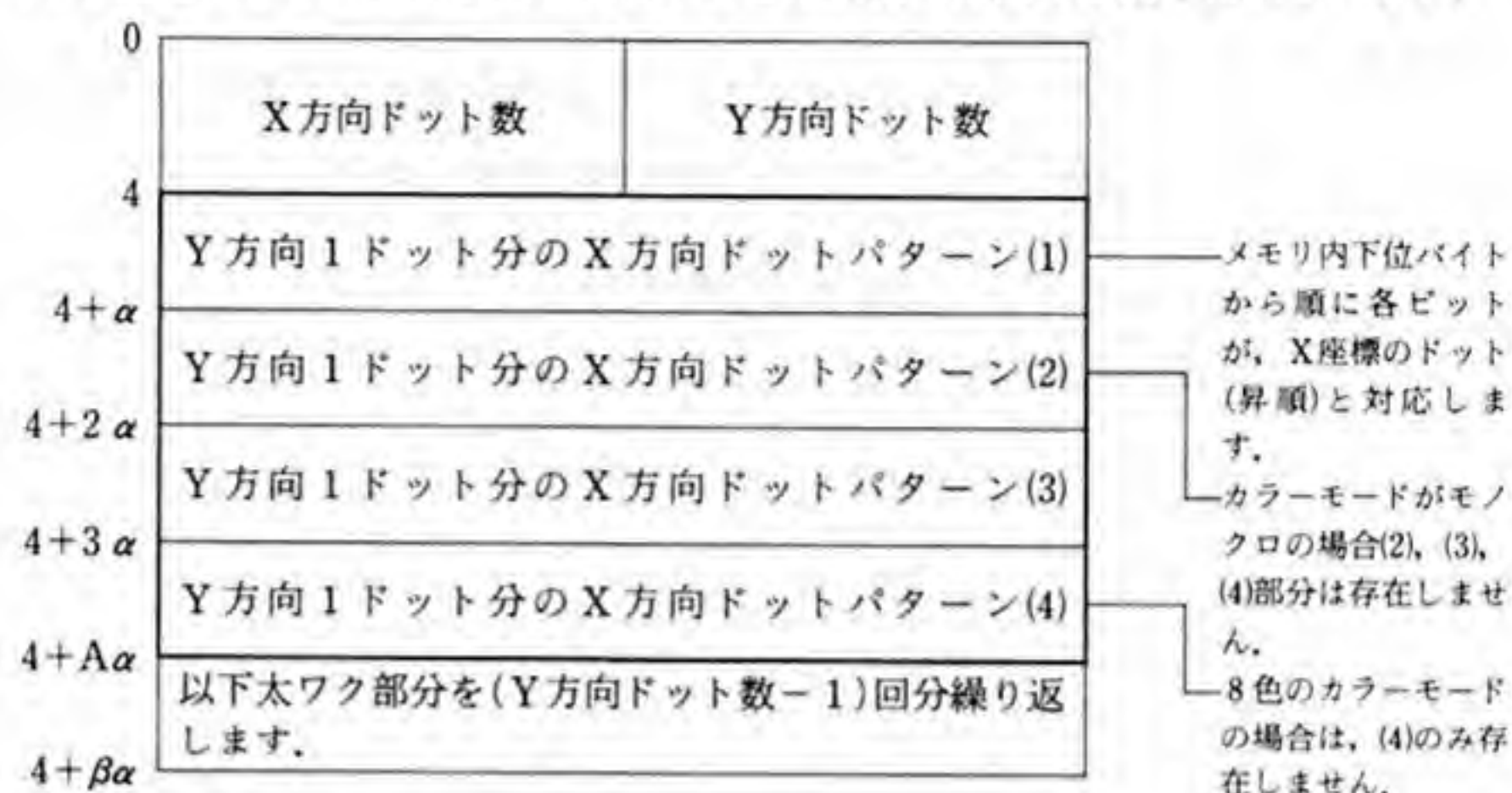
$$\text{格納域長} \geq \{(x2 - x1 + 8) \div 8\} \times (y2 - y1 + 1) \times A + 4$$

ここで、 \div は整数の割り算の商 (余り切り捨て) を意味します。また、A はカラーモードによって次のような値とします。

モノクロ 1
8 色 / 8 色, 8 色 / 4096 色 ... 3
16 色 / 4096 色 4

●格納域のアドレス

メモリ上の格納域のアドレス(上位ワードにセグメント、下位ワードにオフセット)を指定します。矩形領域を取得すると格納域には次の形式で格納されます。



$$\alpha = (x_2 - x_1 + 8) \div 8$$

$$\beta = (y_2 - y_1 + 1) \times A$$

カラーモードがカラーの場合の各ドットのパレット番号は、対応するX方向ドットパターン(1)~(3)(16色モードの場合は(1)~(4))、のビット値(0/1)に、それぞれ1, 2, 4(16色モードの場合はさらに8)を乗算し総和を取ったものです。

カラーモードがモノクロの場合の各ドットの白/黒は、対応するX方向ドットパターン(1)内のビット値が1/0で表わされます。

注意: PC-H98で専用高速描画版グラフィックスドライバを利用する場合、以下の点に留意してください。

- ・256色/1600万色モードではプレーン数の拡張によりグラフィックイメージの格納形式におけるY方向1ドット分のX方向ドットパターンが8個に拡張されます。
- ・16色/1600万色モードは16色/4096色モードのグラフィックイメージの格納形式と同じです。

リターン

AX = 0 ... 正常終了

≠ 0 ... 異常終了(エラーコード一覧参照)

解説

格納域の長さが条件を満たしていないときは、異常終了となって、処理は行われません。一部分が、ビューポート領域からはみ出す場合は、はみ出した部分は、バックグラウンドカラーと見なします。

64 K バイトを超えるグラフィックイメージの取得はできません。

カラーモードがモノクロの場合は、描画プレーンのプレーン0からプレーン3の順でマスクされていない最初のプレーンのグラフィックイメージを格納します。

グラフィックイメージの設定

ファンクション NO. 24

機 能 メモリ上の格納域に取得されたイメージを、システム VRAM 上の指定された 1 点を左上とする矩形領域上に展開します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	予約パラメータ (必ず 0 を指定してください)
4H	WORD	ラストオペレーション番号
6H	WORD	左上 X 座標
8H	WORD	左上 Y 座標
0AH	WORD	メモリ上の格納域の長さ
0CH	DWORD	メモリ上の格納域のアドレス
10H	BYTE	カラースイッチ
12H	DWORD	フォアグラウンドカラー
16H	DWORD	バックグラウンドカラー

ラストオペレーションについては点の描画(ファンクション NO. 15)を参照してください。

格納域の長さ、格納域の形式については、グラフィックイメージの取得(ファンクション NO. 23)を参照してください。

●カラースイッチ

00H : フォアグラウンドカラー/バックグラウンドカラーの指定なし。

(メモリ上のグラフィックイメージは、設定される側の画面モードでグラフィックイメージを取得した場合の格納形式であるものとして、グラフィックイメージの設定を行います。)

01H : フォアグラウンドカラー/バックグラウンドカラーの指定あり。

(メモリ上のグラフィックイメージは、モノクロモードでグラフィックイメージを取得した場合の格納形式であるものとして、グラフィックイメージの設定を行います。)

●フォアグラウンドカラー

モノクロモードで格納されているグラフィックイメージの白(1)のドットを描画するパレット番号を指定します。

●バックグラウンドカラー

モノクロモードで格納されているグラフィックイメージの黒(0)のドットを描画するパレット番号を指定します。

●左上座標

取得されたグラフィックイメージをシステム VRAM に展開する際の矩形の左上頂点を整数系(-32768~32767)座標で指定します。

リターン $AX = 0 \cdots$ 正常終了
 $\neq 0 \cdots$ 異常終了

解説

- ・ビューポート領域からはみ出す部分は描画されません。
- ・表示モードがカラーの場合は、描画プレーンのプレーン0からプレーン3の順でマスクされていないプレーンにグラフィックイメージを展開します。よって、グラフィックイメージを取得したときとグラフィックイメージを設定する場合の表示モード、描画プレーンの情報が異なる場合は、取得前のグラフィックイメージと設定後のグラフィックイメージが異なる可能性があるので注意してください。
- ・表示モードがモノクロの場合は、描画プレーンのプレーン0からプレーン3の順でマスクされていない最初のプレーンにグラフィックイメージを展開します。

領域転送

ファンクション NO. 25

機 能 指定された2点を結ぶ線分を対角線とする矩形領域の内容を、指定された1点を左上とする矩形領域に転送します。システム VRAM、仮想 VRAM のどちらでも転送できます。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	転送元の VRAM
4H	WORD	転送元の矩形領域の左上X座標
6H	WORD	転送元の矩形領域の左上Y座標
8H	WORD	転送元の矩形領域の右下X座標
0AH	WORD	転送元の矩形領域の右下Y座標
0CH	DWORD	転送先の VRAM
10H	WORD	ラストオペレーション番号
12H	WORD	転送先のX座標
14H	WORD	転送先のY座標
16H	WORD	X方向倍率
18H	WORD	Y方向倍率
1AH	BYTE	裏返し
1BH	BYTE	回転

● X 方向倍率/Y 方向倍率

拡大

縮小

0000H = 等倍	8000H = 等倍
0001H = 2 倍	8001H = 1/2 倍
0002H = 4 倍	8002H = 1/4 倍
0003H = 8 倍	8003H = 1/8 倍
0004H = 16倍	8004H = 1/16 倍

●裏返し

転送する際に裏返すかどうかを指定します。裏返しは、転送先の座標を含む、X軸を平行な線分を対象軸として線対象に描画されます。

00H = 裏返しを行わない。

00H ≠ 裏返しを行う。

●回転

転送する際に回転するかどうかを指定します。回転は、転送先の座標を原点にして、反時計回りに 90° 単位で行います。

00H = 回転を行わない。

01H = 90° 回転

02H = 180° 回転

03H = 270° 回転

●転送元の VRAM/転送先の VRAM

システム VRAM を対象とする場合は 0、仮想 VRAM を対象とする場合は仮想 VRAM 構造体の先頭アドレス(上位ワード=セグメント、下位ワード=オフセット)を指定します。

システム VRAM と仮想 VRAM の相互の領域転送も可能です。

●座標

グラフィックイメージの転送元の矩形の左上/右下頂点および転送先の座標を整数系(-32768~32767)座標で指定します。

●ラストオペレーション番号

点の描画(ファンクション NO.15)を参照してください。

リターン

AX = 0 … 正常終了

≠ 0 … 異常終了(エラーコード一覧参照)

解説

- ・転送先のビューポート領域からはみ出す部分は描画されません。
- ・転送元のビューポート領域からはみ出す部分は、バックグラウンドカラーと見なされます。
- ・転送先の描画プレーンに対応する転送元のプレーンが転送されます。そのとき、転送元のプレーンが描画プレーンでない場合は、0 が転送されます。
- ・裏返しと回転を同時に指定した場合は、裏返しを先に行います。
- ・転送元と転送先の矩形領域が同じ VRAM 内で重なるようなときに、拡大、縮小、裏返し、回転を指定した場合の動作は保証できません。

領域移動

ファンクション NO.26

機 能 指定されたドット数分、画面(システム VRAM)上のデータを移動します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	予約パラメータ(必ず0を指定してください)
4H	WORD	ラストオペレーション番号
6H	WORD	X 方向移動ドット数
8H	WORD	Y 方向移動ドット数
0AH	BYTE	クリアフラグ

●ラストオペレーション番号

点の描画(ファンクション NO.15)を参照してください。

●X 方向移動ドット数

この値が正のときは左方向に移動し、負の場合は右方向に移動します。

●Y 方向移動ドット数

この値が正のときは上方向に移動し、負の場合は下方向に移動します。

●クリアフラグ

移動によって、画面に新たに出現する領域を塗りつぶす色を指定します。

クリアフラグ = 0 パレット番号0で塗りつぶします。

クリアフラグ ≠ 0 バックグラウンドカラーで塗りつぶします。

リターン AX = 0 … 正常終了
≠ 0 … 異常終了(エラーコード一覧参照)

解 説 ビューポート領域内のみを移動させます。

バージョンの取得

ファンクション NO.27

機 能 グラフィックスドライバのバージョンを取得します。

コール スタック = データ領域内の先頭アドレス

リターン AX バージョン番号

AL : バージョン番号の整数部

AH : バージョン番号の小数部

解 説 たとえば、バージョン1.0では、AX に 0001H が入ります。

注意：PC-H98 で、専用高速描画版グラフィックスドライバを利用する場合は、
このファンクションで AX=0201H が返されることを確認してください。

プレーン数の取得

ファンクション NO.28

機 能 プレーン数を取得します。

コール スタック = データ領域の先頭アドレス

リターン AH = バンク数
AL = プレーン数

注意：256色オプションボード(PC-H98-E02)を実装した PC-H98 上で専用高速描画版グラフィックスドライバを利用している場合のみプレーン数に8が返されます。

解 説 この機能により、現在の環境で設定できる表示モード、描画プレーン、表示プレーンを知ることができます。

取得 バンク数	取得 プレーン数	ハード ウェア	指定可能 表示モード	指定可能 ページ	指定可能 プレーン
1	3	ノーマル モード	0000 H	0 ~ 1	0 ~ 2
			0001 H	0 ~ 1	0 ~ 2
			0002 H	0 ~ 1	0 ~ 2
			0100 H	0	0 ~ 2
			0101 H	0	0 ~ 2
			0102 H	0	0 ~ 2
	4	ノーマル モード	0000 H	0 ~ 1	0 ~ 3
			0001 H	0 ~ 1	0 ~ 2
			0002 H	0 ~ 1	0 ~ 2
			0003 H	0 ~ 1	0 ~ 3
			0100 H	0	0 ~ 3
			0101 H	0	0 ~ 2
			0102 H	0	0 ~ 2
			0103 H	0	0 ~ 3
		ハイレゾ モード	0300 H	0	0 ~ 3
			0303 H	0	0 ~ 3
			0304 H	0	0 ~ 3
	8	ハイレゾ モード	0300 H	0	0 ~ 7
			0303 H	0	0 ~ 3
			0304 H	0	0 ~ 3
			0305 H	0	0 ~ 7

取得 バンク数	取得 プレーン数	ハード ウェア	指定可能 表示モード	指定可能 ページ	指定可能 プレーン
2	3	ノーマル モード	0000 H	0 ~ 3	0 ~ 2
			0001 H	0 ~ 3	0 ~ 2
			0002 H	0 ~ 3	0 ~ 2
			0100 H	0 ~ 1	0 ~ 2
			0101 H	0 ~ 1	0 ~ 2
			0102 H	0 ~ 1	0 ~ 2
	4	ノーマル モード	0000 H	0 ~ 3	0 ~ 3
			0001 H	0 ~ 3	0 ~ 2
			0002 H	0 ~ 3	0 ~ 2
			0003 H	0 ~ 3	0 ~ 3
			0100 H	0 ~ 1	0 ~ 3
			0101 H	0 ~ 1	0 ~ 2
			0102 H	0 ~ 1	0 ~ 2
			0103 H	0 ~ 1	0 ~ 3
			0104 H	0 ~ 1	0 ~ 3
	8	ノーマル モード	0000 H	0 ~ 3	0 ~ 3
			0001 H	0 ~ 3	0 ~ 2
			0002 H	0 ~ 3	0 ~ 2
			0003 H	0 ~ 3	0 ~ 3
			0100 H	0 ~ 1	0 ~ 7
			0101 H	0 ~ 1	0 ~ 2
			0102 H	0 ~ 1	0 ~ 2
			0103 H	0 ~ 1	0 ~ 3
			0104 H	0 ~ 1	0 ~ 3
			0105 H	0 ~ 1	0 ~ 7

表中の「指定可能表示モード」の値は、表示モードの設定(ファンクション NO. 3) の表示モードとして指定可能な値です。

表中の「指定可能ページ」は、システム VRAM に対する値です。仮想 VRAM に対して指定できるのはページ 0 のみです。

表示モードの取得

ファンクション NO.29

機 能 システム VRAM または仮想 VRAM の現在の表示モードを取得します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	対象 VRAM

●対象 VRAM

システム VRAM の表示モードを取得する場合は 0, 仮想 VRAM の表示モードを取得する場合は仮想 VRAM 構造体の先頭アドレス(上位ワード=セグメント, 下位ワード=オフセット)を指定します。

リターン パラメータブロック

オフセット	サイズ	内 容
4H	WORD	表示モード

AX = 0 … 常に正常終了

解 説 システム VRAM または仮想 VRAM の現在の表示モードを取得します。表示モードについては、表示モードの設定(ファンクション NO.3)を参照してください。

描画プレーンの取得

ファンクション NO.30

機 能 システム VRAM または仮想 VRAM の現在の描画プレーンを取得します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	対象 VRAM

●対象 VRAM

システム VRAM の描画プレーンを取得する場合は 0, 仮想 VRAM の描画プレーンを取得する場合は仮想 VRAM 構造体の先頭アドレス(上位ワード=セグメント, 下位ワード=オフセット)を指定します。

リターン パラメータブロック

オフセット	サイズ	内 容
4H	DWORD	描画プレーン

AX = 0 ... 常に正常終了

解 説 現在の描画プレーンを取得します。描画プレーンについては、描画プレーンの設定(ファンクション NO.4)を参照してください。

表示プレーンの取得

ファンクション NO.31

機 能 現在の表示プレーンを取得します。

コール スタック = データ領域の先頭アドレス

リターン パラメータブロック

オフセット	サイズ	内 容
4H	DWORD	表示プレーン

AX = 0 ... 常に正常終了

解 説 現在の表示プレーンを取得します。表示プレーンについては、表示プレーンの設定(ファンクション NO.5)を参照してください。

パレットの取得

ファンクション NO.32

機 能 指定のパレットに設定されているカラーコードを取得します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
4H	DWORD	パレット番号

リターン パラメータブロック

オフセット	サイズ	内 容
4H	DWORD	カラーコード

AX = 0 … 常に正常終了

解 説 指定されたパレットに設定されているカラーコードを取得します。パレット番号、カラーコードについては、パレットの設定(ファンクション NO.6)を参照してください。

ビューポート領域の取得

ファンクション NO.33

機 能 システム VRAM または仮想 VRAM の現在、設定されているビューポート領域の座標を取得します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	対象 VRAM

●対象 VRAM

システム VRAM のビューポート領域を取得する場合は 0、仮想 VRAM のビューポート領域を取得する場合は仮想 VRAM 構造体の先頭アドレス(上位ワード=セグメント、下位ワード=オフセット)を指定します。

リターン パラメータブロック

オフセット	サイズ	内 容
4H	WORD	ビューポート領域左上 X 座標
6H	WORD	ビューポート領域左上 Y 座標
8H	WORD	ビューポート領域右下 X 座標
0AH	WORD	ビューポート領域右下 Y 座標

AX = 0 … 常に正常終了

解 説 システム VRAM または仮想 VRAM に現在設定されているビューポート領域の、左上点と右下点の座標を取得します。ビューポート領域については、ビューポート領域の設定(ファンクション NO.7)を参照してください。

フォアグラウンドカラーの取得

ファンクション NO.34

機 能 システム VRAM または仮想 VRAM の現在、設定されているフォアグラウンドカラーを取得します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	対象 VRAM

●対象 VRAM

システム VRAM のフォアグラウンドカラーを取得する場合は 0、仮想 VRAM のフォアグラウンドカラーを取得する場合は仮想 VRAM 構造体の先頭アドレス(上位ワード=セグメント、下位ワード=オフセット)を指定します。

リターン パラメータブロック

オフセット	サイズ	内 容
4H	DWORD	フォアグラウンドカラー

AX = 0 … 常に正常終了

解 説 システム VRAM または仮想 VRAM に現在設定されているフォアグラウンドカラーのパレット番号を取得します。フォアグラウンドカラーについては、フォアグラウンドカラーの設定（ファンクション NO.8）を参照してください。

バックグラウンドカラーの取得

ファンクション NO.35

機 能 システム VRAM または仮想 VRAM の現在、設定されているバックグラウンドカラーを取得します。

機 能 スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	対象 VRAM

●対象 VRAM

システム VRAM のバックグラウンドカラーを取得する場合は 0、仮想 VRAM のバックグラウンドカラーを取得する場合は仮想 VRAM 構造体の先頭アドレス(上位ワード=セグメント、下位ワード=オフセット)を指定します。

リターン パラメータブロック

オフセット	サイズ	内 容
4H	DWORD	バックグラウンドカラー

AX = 0 … 常に正常終了

解 説 システム VRAM または仮想 VRAM に現在設定されているバックグラウンドカラーのパレット番号を取得します。バックグラウンドカラーについては、バックグラウンドカラーの設定(ファンクション NO.9)を参照してください。

ボーダーカラーの取得

ファンクション NO.36

機 能 現在、設定されているボーダーカラーを取得します。

コール スタック = データ領域の先頭アドレス

リターン パラメータブロック

オフセット	サイズ	内 容
4H	DWORD	ボーダーカラー

AX = 0 … 常に正常終了

解 説 ボーダーカラーの、現在の設定値を取得します。ボーダーカラーについては、ボーダーカラーの設定(ファンクション NO.10)を参照してください。

表示スイッチの取得

ファンクション NO.37

機 能 現在の表示スイッチの値を取得します。

コール スタック = データ領域の先頭アドレス

リターン パラメータブロック

オフセット	サイズ	内 容
4H	BYTE	表示スイッチ

AX = 0 … 常に正常終了

解 説 表示スイッチの、現在の設定状況を取得します。表示スイッチについては、表示スイッチの設定(ファンクション NO.11)を参照してください。

指定座標のパレットの取得

ファンクション NO.38

機 能 システム VRAM または仮想 VRAM の指定された座標上にある点のパレット番号を取得します。

コール スタック = データ領域の先頭アドレス
パラメータブロック

オフセット	サイズ	内 容
0H	DWORD	対象 VRAM
4H	WORD	取得したい点の X 座標
6H	WORD	取得したい点の Y 座標

●対象 VRAM

システム VRAM の座標のパレット番号を取得する場合は 0、仮想 VRAM の座標のパレット番号を取得する場合は仮想 VRAM 構造体の先頭アドレス(上位ワード=セグメント、下位ワード=オフセット)を指定します。

●X 座標/Y 座標

パレット番号を取得したい点を整数系(−32768〜32767)座標で指定します。

リターン パラメータブロック

オフセット	サイズ	内 容
4H	DWORD	点のパレット番号

AX = 0 … 常に正常終了

解 説 指定の座標がビューポート領域外であれば、点のパレット番号として−1(FFF·FFFFFFH)を取得します。

表示領域の取得

ファンクション NO.39

機 能 システム VRAM 上の表示領域を取得します。

コール スタック = データ領域の先頭アドレス

リターン パラメータブロック

オフセット	サイズ	内 容
4H	WORD	システム VRAM 上の Y 座標

AX = 0 … 常に正常終了

解 説 このファンクションは、ハイレゾリレーションモードでのみ意味を持ち、VRAM 上の表示開始 Y 座標を取得します。表示領域については、表示領域の設定 (ファンクション NO.12) を参照してください。

ノーマルモードでは、常に 0 を取得します。

中断処理ルーチンの取得

ファンクション NO.40

機 能 現在、設定されている中断処理ルーチンの先頭アドレスを取得します。

コール スタック = データ領域の先頭アドレス

リターン パラメータブロック

オフセット	サイズ	内 容
4H	DWORD	中断処理ルーチンのアドレス

AX = 0 … 常に正常終了

解 説 現在、設定されている中断処理ルーチンの先頭アドレスを取得します。アドレスは上位ワードがセグメント、下位ワードがオフセットです。中断処理ルーチンの設定(ファンクション NO.13)を参照してください。

4.5 エラーコード一覧

ここでは、エラーコードについて説明します。

各ファンクションでは、通常、リターン値が $AX \neq 0$ のときは、その値はエラーコードを示します。以下にエラーコードとその意味について示します。

エラーコード	意 味
01	不正呼び出し。パラメータに誤りがあります。
02	作業域不足。上位から受け取った作業域のサイズでは、機能が実行できません。
03	演算結果のオーバーフロー。楕円描画などで不正なパラメータにより描画の実行が不可能です。
04	不正ファンクション。現在このファンクションは定義されていません。
05	データ領域のアドレス不正。EMS 利用時に不正なアドレスがパラメータに指定されているため、機能の実行が不可能です。EMS 領域内のアドレスが指定された場合などにエラーとなります。

注意：上記のエラーコードのうち、05 は PC-H98 にて専用高速描画版グラフィックスドライバを使用している場合のみ返されるものです。

4.6 専用高速描画版 (GRP_H98.LIB) と GRAPH.LIB の互換性について

PC-H98 で専用高速描画版グラフィックスドライバ (GRP_H98.LIB) を組み込んだ場合、使用するグラフィックス用ハードウェアの特性により、一部の描画ファンクションで描画結果が非互換となります。詳細は次の表を参照してください。

項番	描画ファンクション	描画結果が GRAPH. LIB と異なる点	条 件	備 考
1	三角形の描画	輪郭を構成する点の位置が1ドットずれることがあります。	線種パターンと線幅を省略し、輪郭と同じ色で内部を塗りつぶす場合	線種パターンを32ビットで指定すると描画結果は GRAPH. LIB と同じになります。ただし、処理速度も GRAPH. LIB と同等になります。
2	台形の描画	同上	線種パターンと線幅を省略し、輪郭と同じ色で内部を塗りつぶす場合	
3	円の描画	扇形の終了線分を構成する点の位置が1ドットずれることがあります。 また、ラストオペレーションで0以外の値を指定すると扇形の弧と開始線分と終了線分の交点で描画色が異なります。	線種パターンを省略し、塗りつぶしを行わない扇形の描画	
4	楕円の描画	周と扇形の終了線分を構成する点の位置が1ドットずれることがあります。 また、ラストオペレーションで0以外の値を指定すると扇形の弧と開始線分と終了線分の交点で描画色が異なります。	・線種パターンを省略した次の楕円の描画(楕円、弧、扇形) ・輪郭と同じ色で内部を塗りつぶす楕円	
5	領域転送	縮小するときはドットの間引き方により転送結果が異なります。 また、拡大するときは転送開始位置が(拡大率-1)ドット異なります。	転送元と転送先が共にシステム VRAM であり拡大または縮小を行なう場合	仮想 VRAM を経由してシステム VRAM に転送すると描画結果は GRAPH. LIB と同じになります。ただし、処理速度は低下します。

第5章

EMSインターフェイス

5.1 イントロダクション

アプリケーションプログラムによっては、通常のメモリ容量の最大値(640 K バイト)よりも大きなメモリ領域を必要とするものがあります。このような場合のために、拡張メモリ(1 メガバイト以上のメモリ空間)をアプリケーションプログラムで使えるようにしたものが、拡張メモリ仕様(EMS)です。

拡張メモリ仕様(EMS)は、“拡張メモリマネージャ”(略号“EMM”)と呼ぶ、拡張メモリを制御し管理するデバイスドライバと、拡張メモリを使用するアプリケーションプログラムとのインターフェイスからなるソフトウェアです。

拡張メモリとは何か

拡張メモリは、MS-DOSの640Kバイト(ハイレゾリューションモードでは768Kバイト)の制限を超えたメモリです。EMSは、32メガバイトまでの拡張メモリの増設をサポートします。V30, 8086, 8088, 80286, 386/386SX (リアルモード)のマイクロプロセッサ(CPU)は、物理的に1メガバイトのアドレスしか持つことができませんが、その様な物理アドレスの範囲にある“ウィンドウ”を通じて拡張メモリにアクセスすることができます。次節では、これをどのように実現するかを説明しています。

拡張メモリの働き

拡張メモリは、“論理ページ”と呼ばれるメモリのブロックに分割されます。ひとつの論理ページの、典型的な大きさは16 K バイトです。コンピュータは、“ページフレーム”と呼ばれる物理的なメモリのブロックを通して論理ページにアクセスします。ページフレームは、多数の物理ページ、すなわちマイクロプロセッサが直接的にアドレスを付けることができるページを含みます。物理ページの典型的な大きさも、16 K バイトです。

このページフレームは、拡張メモリに対して“ウィンドウ”として機能します。ちょうど、コンピュータの画面がスプレッドシートに対するウィンドウであるように、ページフレームは、拡張メモリに対するウィンドウになっています。

拡張メモリの論理ページは、ページフレームの、ある物理ページにマッピングされ(存在するかのように作られ)ます。それゆえ、実際の物理ページに対する読み込みや書き出しは、関連する論理ページへの読み込みや書き出しになります。ひとつの論理ページは、ある物理ページに対するページフレームにマッピングされます。

図1-1は、ページフレーム、物理ページ、論理ページの関係を示しています。

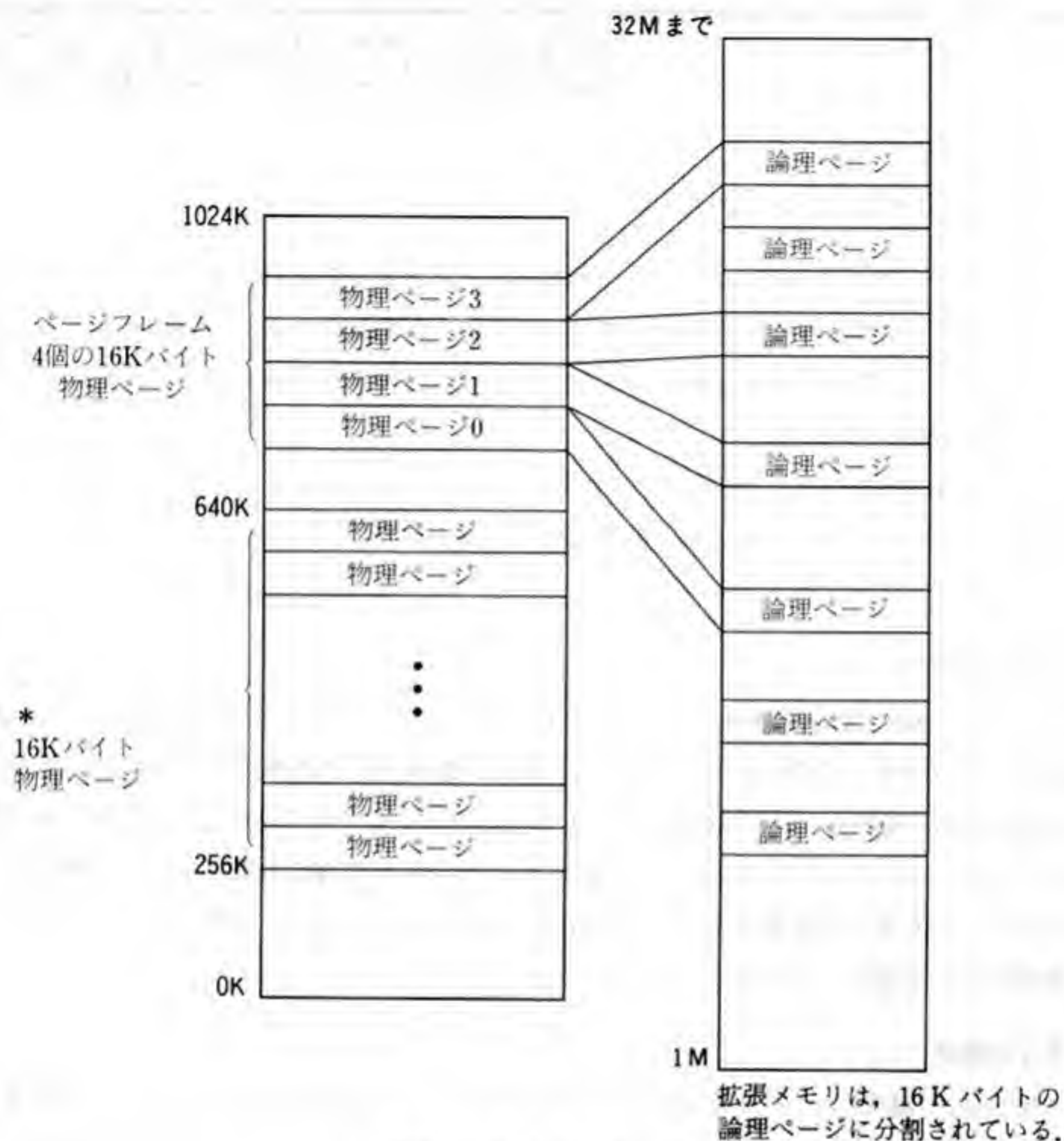


図1-1 拡張メモリ

*オペレーティングシステムに限って使用できることを意味している。

*この図は、概念的なものであり、PC-9800シリーズのEMSドライバでは、640 K バイトよりも下位のアドレスにはページフレームがありません。

ページフレームは、640 K バイトよりも上位のアドレスに位置しています。通常 640 K～1024 Kの間には、VRAM や拡張 ROM 空間があります。

EMS では、オペレーティングシステムに対して、640 K バイトよりも下位のアドレスにある物理ページを通じて拡張メモリにアクセスする方法も定義しています（ただし、PC-9800 シリーズの EMS ドライバでは使用できません）。これらの方法は、オペレーティングシステムを発展させるものとしてのみ意味を持ちます。

PC-9800 シリーズでは、ページフレームのアドレス（位置）とサイズは、機種やハードウェアモードによって異なります。次に機種ごとの、ページフレームのアドレスと、ページフレームのサイズを示します。

ハードウェアモード	ページフレームアドレス	物理ページ数
ノーマルモード機種 ^(注1)	B0000H～BFFFFH ^(注2)	4 ページ
	C0000H～CFFFFH ^(注3)	
	C0000H～C7FFFH ^(注3)	2 ページ
	C8000H～CFFFFH ^(注3)	
ハイレゾリューションモード	B0000H～BFFFFH	4 ページ

(注1) ページフレームアドレスは、使用機種、使用する EMS ドライバによって異なります。詳しくは「MS-DOS 3.3 B ユーザーズリファレンスマニュアル」を参照してください。

(注2) B0000 H～BFFFFH は、グラフィック VRAM と重なる (EMS 用ページフレームとバンク切り換えによって使用している) ため、アプリケーションプログラム自身が、ページフレームと VRAM を切り換えて使用する必要があります。

この VRAM とページフレームの切り換えは、EMS ドライバを呼び出すことによって行うことができます。詳しくは、5.4.2 の EMS ドライバファンクション 31 の説明を参照してください。

(注3) EMS 機能付き増設 RAM ボードが必要です。

5.2 拡張メモリを使用するプログラムの書き方

ここでは、拡張メモリを使用するすべてのプログラムが行わなければならないことや、拡張メモリを利用するための、より高度なテクニックを説明します。

5.2.1 すべてのプログラムが実行しなければならないこと

ここでは、拡張メモリを使用するために、すべてのプログラムが実行しなければならないことについて説明します。

拡張メモリを使用するために、アプリケーションプログラムでは、次のような手続きを実行しなければなりません。

1. EMM がインストールされているかどうかを確認する。
2. 拡張メモリのページが、アプリケーションに対して充分なだけ存在するかどうか判断する (ファンクション 3)。
3. 拡張メモリのページをアロケートする (ファンクション 4 または 18)。
4. 使用する各物理ページの基底アドレスを求める (ファンクション 25)。
5. 拡張メモリを物理ページにマッピングする (ファンクション 5 または 17)。
6. 物理ページを標準メモリと同じようにアクセスし、拡張メモリのデータの読み込み／書き出し／実行を行う。
7. 拡張メモリの使用を終える前に、拡張メモリのページをデアロケートする (ファンクション 6 または 18)。

次の表は、1～7 の各続きで使用する基本ファンクションの機能の概略です。各ファンクションの詳しい説明は、5.4 EMS ファンクションリクエストを参照してください。

基本ファンクション一覧

手続き ファンクションと機能

-
- | | |
|---|--|
| 1 | Get Status
メモリ管理ハードウェアが、正しく動作しているかどうかを示すステータスコード値を返す。 |
| 2 | Get Page Frame Address
ページフレームが位置しているアドレスを返す。 |
| 3 | Get Unallocated Page Count
プログラムで利用できる、まだアロケートされていない論理ページ数と、拡張メモリの全論理ページ数の値を返す。 |
| 4 | Allocate Pages
要求された数の論理ページをアロケートし、これらの論理ページに固有の EMM ハンドルを割り当てる。 |

5 Map/Unmap Handle Page

論理ページを物理ページにマッピングする。

6 Deallocate Pages

EMM ハンドルに割り当てられている論理ページを、デアロケートする。

7 Get Version

EMSインターフェイスのバージョン値を返す。

5.2.2 応用プログラミング

EMS は、基本ファンクションに加えて、拡張メモリを使用するソフトウェアの能力を高める、いくつかの応用ファンクションを備えています。

以下では、応用プログラミングと、それらのプログラミングで利用するファンクションを説明します。

注意：プログラムで応用ファンクションを使用する前に、まずインストールされている EMM が、これらのファンクションをサポートしているバージョンであるかどうかを調べてください。これには、ファンクション 7 (Get Version ファンクション) を使用します。

5.2.3 物理ページのマッピングの状態を保存する

割り込みサービスルーチン、デバイスドライバ、常駐ソフトウェアのようなソフトウェアは、次のような処理を行わなければなりません。

- 物理ページの現在のマッピング状態の保存
- マッピングコンテキストの切り換え
- 拡張メモリの区域の操作
- 物理ページのマッピング状態の復元

物理ページの状態を保存するには、ファンクションの 8 と 9、またはファンクションの 15 と 16 を使用します。

5.2.4 ハンドルの検索とページカウント

ある種のユーティリティプログラムでは、どのような方法で拡張メモリが使用されつつあるかを記録した軌跡を保持する必要があります。これを実行するには、ファンクション 12~14 を使用します。

5.2.5 複数ページのマッピングとアンマッピング

複数ページのマッピングは、あるアプリケーションのマッピング中にかかるオーバーヘッドを減少させます。複数ページのマッピングとアンマッピングには、ファンクション 17 を使用します。

さらに、物理ページの代わりに、セグメントアドレスを使用してマッピングすることもできます。たとえば、ページフレームの基底アドレスが C 000 H にセットされていれば、物理ページ 0 かセグメント C 000 H へマップすることができます。すべての拡張メモリの物理ページと、それらに対応するセグメント値のクロスリファレンスを得るには、ファンクション 25 (Get Mappable Physical Address Array ファンクション) を使用します。

5.2.6 ページのリアロケート

ファンクション 18 (Reallocate Pages) を使用すると、EMM ハンドルに割り当てられている論理ページ数を変更することができます。

5.2.7 ハンドルの使用とハンドルへの名前の割り当て

EMM ハンドルには 8 文字 (8 バイト) の名前であるハンドル名を付けることができます。

ハンドル名を付けることにより、複数のアプリケーションプログラムで使用する共通の拡張メモリを設けることができます。

例えば、複数のアプリケーションプログラムで共通に使用する EMM ハンドルにハンドル名を付けて、共通の EMM ハンドルにします。そして EMM は、共通の EMM ハンドルをハンドル名で探し、探し出した EMM ハンドルの論理ページをアクセスすることで、共通の拡張メモリが設けられます。

ハンドル名を EMM ハンドルに付けるには、ファンクション 20 (Set Handle Name サブファンクション) を使用します。また、特定のハンドル名が付けられている EMM ハンドルを得るには、ファンクション 21 (Search for Named Handle サブファンクション) を使用します。さらに、EMM ハンドルに割り当てられた論理ページ数を調べるには、ファンクション 14 (Get Handle Pages ファンクション) を使用します。

5.2.8 ハンドルの属性の使用

EMM ハンドルには名前をつけるだけでなく、属性 (揮発性 / 不揮発性) を付けることができます。これには、ファンクション 19 を使用します。

属性が不揮発性とは、拡張メモリのデータはウォームブート保持することができる属性のことです。逆に、属性が揮発性では、ウォームブートの前のデータはウォームブート後は保持されません。ハンドルの初期属性は、揮発性になっています。

このファンクションの利用は、そのシステムに組み込まれた拡張メモリのハードウェアに依存しています。したがって、ハンドルのページに属性を割り当てる前に、ファンクション 19 (Get Attribute Capability サブファンクション) を使用し、ハードウェアの属性を調べる必要があります。

5.2.9 ページマップの変更と飛び越し／呼び出し

拡張メモリに格納されているプログラムを実行するには、ファンクション 22 (Alter Page Map and Jump) または、ファンクション 23 (Alter Page Map and Call) を使用します。

このファンクションは、プログラムが格納されている拡張メモリの論理ページをマッピングし、ジャンプまたはコールを行います。

拡張メモリにプログラムを格納することにより、標準メモリの容量を超えるプログラムを実行することができたり、使用する標準メモリの容量を少なくすることができます。

5.2.10 メモリ領域の移動と交換

ファンクション 24 (Move/Exchange Memory Region ファンクション) を使用すると、標準メモリと拡張メモリの間で、簡単にデータ移動／交換することができます。ファンクション 24 は、1 回のファンクション呼び出しで、1 メガバイトまでのデータを扱うことができます。

アプリケーションは、このファンクションがなくても、同じ処理を実行することができますが、拡張メモリマネージャを使用するとアプリケーションのオーバーヘッドが減少します。

5.2.11 物理ページの数と各物理ページのアドレスを得る

ファンクション 25 を使用すると、物理ページの数と各物理ページのアドレスを調べることができます。サブファンクションとして、物理ページの数と各物理ページのアドレスを返すものがあります。

5.2.12 オペレーティングシステムファンクション

アプリケーションプログラムに対するファンクションに加えて、EMS ではオペレーティングシステムに対するファンクションも定義しています。これらのファンクションは、オペレーティングシステムによっては、無効にされる場合もあります。したがって、プログラムがこれらのファンクションに過度に依存することは、望ましいことではありません。この警告を無視して、このファンクションを使用するアプリケーション（オペレーティングシステムも含む）は、他のプログラムと互換性のないものになる危険性が高くなります。

応用ファンクション一覧

8 Save Page Map

拡張メモリマネージャ内部の記憶領域に、物理ページのマッピング状態を保存します。

9 Restore Page Map

物理ページのマッピング状態を、拡張メモリマネージャ内部の記憶領域に保存されている状態に戻します。

10 (システム予約)

- 11 (システム予約)
- 12 Get Handle Count
使用している EMM ハンドルの数を返します。
- 13 Get Handle Pages
EMM ハンドルに割り当てられている論理ページ数を返します。
- 14 Get All Handle Pages
使用しているすべての EMM ハンドルの値と、それぞれの EMM ハンドルに割り当てられているページ数を返します。
- 15 Get/Set Page Map
指定したメモリ領域に、すべての物理ページのマッピング状態を保存、および保存した状態に復元します。
- 16 Get/Set Partial Page Map
指定したメモリ領域に、指定した物理ページのマッピング状態を保存、および保存した状態に復元します。
- 17 Map/Unmap Multiple Handle Pages
複数の物理ページのマッピングを一度に行います。
- 18 Reallocate Page
EMM ハンドルに割り当てられている論理ページ数を変更します。
- 19 Get/Set Handle Attribute
EMM ハンドルの属性を取得/設定します。
- 20 Get/Set Handle Name
EMM ハンドルに付けられている名前の取得、およびハンドルへ名前を付けます。
- 21 Get Handle Directory
使用中の EMM ハンドルについての情報と、それぞれに割り当てられた名前を返します。
- 22 Alter Page Map and Jump
論理ページをマッピングして、指定したアドレスへジャンプします。
- 23 Alter Page Map and Call
論理ページをマッピングして、指定したアドレスをコールします。制御が呼び出し側に戻るときも、論理ページをマッピングします。
- 24 Move/Exchange Memory Region
メモリ領域を、標準メモリから標準メモリへ、標準メモリから拡張メモリへ、拡張メモリから標準メモリへ、拡張メモリから拡張メモリへ、コピーや交換を行います。
- 25 Get Mappable Physical Address Array
すべての物理ページのセグメントアドレスと、物理ページ数を返します。

- 26 Get Expanded Memory Hardware Information
拡張メモリのハードウェア情報を返します。
- 27 Allocate Standard/Raw Pages
オペレーティングシステムが使用する論理ページをアロケートします。
- 28 Alternate Map Register Set
マルチタスキングシステムにおける、物理ページのマッピング状態を制御します。
- 29 Prepare Expanded Memory Hardware for Warm Boot
ウォームブートに対する、拡張メモリの準備を行います。
- 30 Enable/Disable OS/E
オペレーティングシステム開発者が、オペレーティングシステムが使用するファンクションを有効にしたり、無効にしたりします。

5.3 プログラミングのガイドライン

ここでは、EMSインターフェイスを使用するアプリケーションプログラムを開発する際の、ガイドラインを解説します。

- 拡張メモリをプログラムスタックに使用してはいけません。
- 割り込み 67 H を変更してはいけません。これは EMS インターフェイス使用時の割り込みベクトルです。割り込み 67 H を変更すると、EMS インターフェイスが使用できなくなります。
- アプリケーションプログラムで、標準メモリにある物理ページを使用してはいけません。
もし、使用する場合は、アプリケーションプログラムが標準メモリにある物理ページのメモリ空間を確保しなければなりません。確保しないで使用すると、プログラムの暴走の原因となります。
- 同じ EMM ハンドルの同じ論理ページを複数の物理ページにマッピングする場合に注意が必要です。

この場合、物理ページにデータを書き込むと、他の物理ページにマッピングされている同じ論理ページにも、自動的にデータが書き込まれる場合と、書き込まれない場合があります。どちらの状態であるかは、同じ論理ページを別々の物理ページにマッピングし、片方の物理ページにデータを書き込み、もう片方の物理ページにも書き込まれているかで確認します。

これらは、拡張メモリのデータエイリアシング（同じ論理ページを別々の領域として扱うこと）を行う場合に注意しなければなりません。

●アプリケーションは、終了時に必ず EMM ハンドルをデアロケートしなければなりません。これらのデアロケートした論理ページは、他のアプリケーションで使えるようになります。もし、デアロケートしないと、その論理ページは EMM ハンドルにアロケートされたままになり、他のアプリケーションプログラムで使えなくなります。

また、このようなことが繰り返し起こると、論理ページまたは EMM ハンドルを使い果たすことになり EMS インターフェイスが使えなくなります。

● Terminate and Stay Resident Programs (TSR's, 終了後もメモリに残るプログラムで、常駐プログラムともいう) は、論理ページのマッピングを行う前に、必ず物理ページのマッピング状態を保存するようにします。TSR's が、拡張メモリを使用している可能性のある、他のプログラムに割り込みをかける場合があるので、最初に物理ページのマッピング状態を保存しないで、論理ページをマッピングしてはいけません。また、終了する前には、TSR's は、物理ページのマッピング状態を元に戻さなければなりません。

次に、物理ページのマッピング状態を保存し、復元する 3 つの方法を説明します。

1. Save Page Map ファンクション (ファンクション 8) と Restore Page Map ファンクション (ファンクション 9)

これは、3 つの方法の中で、いちばん簡単な方法です。ファンクション 8 (Save Page Map) で物理ページの状態を保存し、ファンクション 9 (Restore Page Map) で物理ページの状態を復元します。物理ページの状態を保存するメモリ領域は EMM の内部にあるのでアプリケーションプログラムで用意する必要はありません。

物理ページの状態を復元するとき、ファンクション 8 (Save Page Map) で使用した EMM ハンドルをファンクション 9 (Restore Page Map) で指定しないと、正しく復元されませんので注意してください。

この方法は、一度保存すると復元するまで同じ EMM ハンドルを使用して保存することができます。復元を行うと、再度同じ EMM ハンドルに保存することができます。

また、保存される物理ページの状態は、EMS 標準 (64 K バイト) のページフレーム内にある物理ページのマッピング状態のみです。

2. Get/Set Page Map ファンクション (ファンクション 15)

この方法は、アプリケーションプログラムが用意したメモリ領域に物理ページのマッピング状態を保存します。この方法で物理ページ状態を復元するには、アプリケーションプログラムが以前に物理ページ状態を保存した、メモリ領域のアドレスを指定します。

もし、アプリケーションプログラムが復元の前に 1 回以上の、物理ページのマッピング状態を保存する必要がある場合は、この方法を使用します。

3. Get/Set partial Page Map ファンクション (ファンクション 16)

この方法は、指定した物理ページのマッピング状態を保存する方法です。アプリケーションプログラムがすべての物理ページのマッピング状態を保存する必要がないとき、この方法を使用します。このファンクションでは、保存するメモリ領域をアプリケーションプログラムで用意してください。

●アプリケーションプログラムで用意する保存領域は論理ページをマッピングする物理ページ上に設けてはいけません。Alter Map and Jump ファンクション (ファンクション 22) と Alter Map and Call ファンクション (ファンクション 23) は例外です。

5.4 EMS ファンクションリクエスト

5.4.1 ファンクションリクエスト一覧

次に、EMS インターフェイスで使用できる、ファンクションリクエストの一覧を示します。各ファンクションリクエストは、5.4.2 EMS ファンクションリクエストで詳細に説明されています。

ファンクションリクエスト一覧

① 基本ファンクション

番号	コール	ファンクション名
01	40H	ステータスの取得
02	41H	ページフレームのアドレス取得
03	42H	未アロケートページカウントの取得
04	43H	ページのアロケート
05	44H	ハンドルページのマップ/アンマップ
06	45H	ページのデアロケート (解放)
07	46H	バージョンの取得

② 拡張ファンクション

番号	コール	ファンクション名
08	47H	ページマップのセーブ
09	48H	ページマップのリストア
12	4BH	ハンドルのカウントの取得
13	4CH	ハンドルページの取得
14	4DH	全ハンドルページの取得
15	4E00H	ページマップの取得/セット (ページマップの取得 サブファンクション)
15	4E01H	ページマップの取得/セット (ページマップセット サブファンクション)
15	4E02H	ページマップの取得/セット (ページマップの取得/セット サブファンクション)
15	4E03H	ページマップの取得/セット (ページマップのセーブアレイのサイズ取得サブファンクション)
16	4F00H	ページマップの一部の取得/セット (ページマップの一部取得 サブファンクション)
16	4F01H	ページマップの一部の取得/セット (ページマップの一部セット サブファンクション)
16	4F02H	ページマップの一部の取得/セット (ページマップセーブアレイの一部のサイズ取得サブファンクション)
17	5000H	複数のハンドルページのマップ/アンマップ (論理ページ/物理ページ 方式)

17	5001H	複数のハンドルページのマップ/アンマップ (論理ページ/セグメント アドレス方式)
18	51H	ページの再アロケート
19	5200H	ハンドルアトリビュートの取得/セット (ハンドルアトリビュートの取得)
19	5201H	ハンドルアトリビュートの取得/セット (ハンドルアトリビュートのセット)
19	5202H	ハンドルアトリビュートの取得/セット (アトリビュートのケイパビリティの取得)
20	5300H	ハンドル名の取得/セット (ハンドル名の取得)
20	5301H	ハンドル名の取得/セット (ハンドル名のセット)
21	5400H	ハンドルのディレクトリ取得 (ハンドルのディレクトリ取得)
21	5401H	ハンドルのディレクトリ取得 (名前よりハンドルのサーチ)
21	5402H	ハンドルのディレクトリ取得 (ハンドルの総数の取得)
22	55H	ページマップの変更とジャンプ
23	56H	ページマップの変更とコール (ページマップの変更とコール)
23	5602H	ページマップの変更とコール (ページマップ スタックサイズの取得)
24	5700H	メモリ領域の移動/交換 (メモリ領域の移動)
24	5701H	メモリ領域の移動/交換 (メモリ領域の交換)
25	5800H	マップ可能な物理アドレスアレイの取得 (マップ可能な物理アドレスアレイの取得)
25	5801H	マップ可能な物理アドレスアレイの取得 (マップ可能な物理アドレスアレイエントリの取得)
26	*	拡張メモリハードウェア情報の取得 (ハードウェア構成のアレイ取得)
26	*	拡張メモリハードウェア情報の取得 (未アロケートのローページカウンタの取得)
27	5AH	ローページのアロケート
28	*	マップレジスタの変更
28	*	マップレジスタセットの変更 (変更マップレジスタセットの取得)
28	*	マップレジスタセットの変更 (変更マップセーブアレイのサイズ取得)

注：*はOSのみが使用可能なファンクションです。

番号	コール	ファンクション名
28	*	マップレジスタの変更 (変更マップレジスタセットのアロケート)
28	*	マップレジスタセットの変更 (変更マップレジスタセットのデアロケート)
28	*	マップレジスタの変更 (DMA レジスタセットのアロケート)
28	*	マップレジスタの変更 (変更マップレジスタ上の DMA のイネーブル)
28	*	マップレジスタの変更 (変更マップレジスタ上の DMA のディセーブル)
28	*	マップレジスタの変更 (DMA レジスタセットのデアロケート)
29	5CH	ウォームブートのために拡張メモリハードウェアを準備
30	*	OS/E ファンクションセットのイネーブル/ディセーブル (OS/E ファンクションセットのイネーブル)
30	*	OS/E ファンクションセットのイネーブル/ディセーブル (OS/E ファンクションセットのディセーブル)
30	*	OS/E ファンクションセットのイネーブル/ディセーブル (アクセスキーのリターン)
31	7000H	ページフレーム用バンクのステータスを取得する
31	7001H	ページフレーム用バンクの有効/無効の切換え

注：*は OS のみで使用可能なファンクションです。

5.4.2 EMS ファンクションリクエスト

次に、EMS インターフェイスで使用できるファンクションリクエストを、コード番号順に説明します。

INT 67H

Get Status

ファンクション No. 1

機能 ステータスの取得

コール AH=40H

リターン AH = 00H 正常実行 (メモリマネージャあり, ハードウェアは正常に動作)
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)

解説 メモリマネージャが存在し, ハードウェアが正しく動作しているかどうかのチェックを行います。

例

MOV	AH,40H	; load function code
INT	67H	; call the memory manager
OR	AH,AH	; check EMM status
JNZ	EMM_ERR_HANDLER	; jump to error handler on error

INT 67H

Get Page Frame Address

ファンクション No. 2

- 機能** ページフレームアドレスの取得
- コール** AH = 41H
- リターン** AH = 00H 正常実行 (ページフレームアドレスが BX にセットされた)
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)
 BX ページフレームセグメントアドレス (AH に 0 が返ったときのみ)
- 解説** ページフレームが位置しているセグメントアドレスを返します。
- 例**
- ```

page_frame_segment DW ?
MOV AH,41H ; load function code
INT 67H ; call the memory manager
OR AH,AH ; check EMM status
JNZ EMM_ERR_HANDLER ; jump to error handler on error
MOV page_frame_segment,BX ; save page frame address

```



## INT 67H

## Get Unallocated Page Count

## ファンクション No. 3

**機 能** 未アロケートページカウン트의取得

**コール** AH = 42H

**リターン** AH = 00H 正常実行 (未アロケートページ数と総ページ数が BX と DX に返された)

= 80H 回復不可 (メモリマネージャソフトウェアが動作しない)

= 81H 回復不可 (拡張メモリハードウェアが動作しない)

= 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)

BX 未アロケートページ数

DX 総ページ数

**解 説** 未アロケートページ数と、拡張メモリページの総数を返します。

|          |                       |                                  |
|----------|-----------------------|----------------------------------|
| <b>例</b> | un_alloc_pages        | DW ?                             |
|          | total_pages           | DW ?                             |
|          | MOV AH,42H            | ; load function code             |
|          | INT 67H               | ; call the memory manager        |
|          | OR AH,AH              | ; check EMM status               |
|          | JNZ EMM_ERR_HANDLER   | ; jump to error handler on error |
|          | MOV un_alloc_pages,BX | ; save unallocated page count    |
|          | MOV total_pages,DX    | ; save total page count          |

## INT 67H

### Allocate Pages

### ファンクション No. 4

- 機能** ページのアロケート
- コール** AH = 43H  
BX = アロケートしたいページ数
- リターン** AH = 00H 正常実行 (ページがアロケートされた)  
= 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
= 81H 回復不可 (拡張メモリハードウェアが動作しない)  
= 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
= 85H 復旧可能 (全 EMM ハンドルは使用中)  
= 87H 復旧可能 (プログラムの要求を満たすような拡張メモリページがシステム内に存在しない)  
= 88H 復旧可能 (プログラムの要求を満たすような未アロケートページがない)  
= 89H 復旧可能 (プログラムが 0 ページをアロケートしようとした)

#### DX EMM ハンドル

- 解説** 要求されたページ数をアロケートし、アロケートしたページに EMM ハンドルを割り付けます。その EMM ハンドルはアプリケーションがページを解放するまで、そのページを保有します。

このファンクションを使用して割り付けられたハンドルは、16KB のページを所有します。このサイズは拡張メモリの標準サイズです。拡張メモリボードハードウェアが、16KB サイズのページの供給ができない場合は、標準サイズではないページをいくつか結合して 16KB ページの形になるようにします。

EMM が返すハンドルの数値は、10 進の 1~254 (0001H~00FEH) の範囲です。

OS ハンドル (ハンドル 0) が、このファンクションで返されることはありません。また、ハンドルの最上位バイトは 00H であり、アプリケーションプログラムで使用することはできません。メモリマネージャは 255 (ハンドル 0 を含む) までのハンドルを供給でき、EMM がいくつかのハンドルをサポートしているかは、ファンクション 21 を使用して知ることができます。

このファンクションでは、ハンドルに 0 ページをアロケートすることはできません。アプリケーションプログラムが 0 ページをアロケートする場合は、ファンクション 27 を使用します。

#### ●パラメータの説明

DX= EMM ハンドル

以降このハンドルを使用する。ハンドルは 255 まで。ハンドルの最上位バイトは 0 で、アプリケーションプログラムでは使用不可。

#### 注 意

次の注意事項は、拡張メモリマネージャインプリメンタと OS 開発者のみに関係するものです。アプリケーションプログラムのユーザーは、このメモリマネージャの特質を利用することはできません。

拡張メモリマネージャには、OS のみが使用可能なハンドルが準備されています。このハンドルは 0000H の値で、拡張メモリマネージャがインストールしたときに、そのハンドルにアロケートされているページの 1 セットを所有します。メモリマネージャが 0000H のハンドルに自動的にアロケートするページは、実メモリの上位アドレスに存在し、一般的にこれは 40000H (256K) から 9FFFFH (640K) アドレス間に存在します。しかし、この範囲は、ハードウェアやメモリマネージャをサポートしているなら、この範囲を上下に拡張することができます。

OS は、このハンドルを得るためにファンクション 4 を呼び出す必要はありません。拡張メモリデバイスドライバがインストールされると、このハンドルは既に存在していると想定され、ただちに使用可能になるからです。

OS がハンドルを使用する場合は、0000H の特別なハンドル値を使用します。これにより OS はどんな EMM ファンクションでも呼び出すことができます。また、このハンドルにページをアロケートするためにはファンクション 18 (ページの再アロケート) を呼びびします。

このハンドルには、2 つの特別な場合があります。

##### 1. ファンクション 4 (Allocate Pages ファンクション)

このファンクションは、ハンドル値としてゼロを返すことはありません。アプリケーションにおいては、ページをアロケートし、そのページをもつハンドルを得るためにはファンクション 4 を呼び出さなければなりません。ファンクション 4 はゼロのハンドル値を返すことはないので、アプリケーションでこの特別なハンドルにアクセスすることはできません。



## 2. ファンクション 6 (Deallocate Pages ファンクション)

オペレーティングシステムが特別な EMM ハンドルにアロケートされたページの解除のためにこのファンクションを使用すると、EMM ハンドルが所有するページは使用可能としてメモリマネージャに返されます。しかしこの EMM ハンドルは再び割り当てすることはできません。メモリマネージャは Deallocate Pages ファンクションのこの EMM ハンドルへの要求を、Reallocate Pages ファンクションの要求と同じものとして取り扱います。すなわち、この EMM ハンドルへのリアロケートのページ番号はゼロになります。

|   |                              |                                  |
|---|------------------------------|----------------------------------|
| 例 | num_of_pages_to_alloc        | DW ?                             |
|   | emm_handle                   | DW ?                             |
|   | MOV BX,num_of_pages_to_alloc | ; load number of pages needed    |
|   | MOV AH,43H                   | ; load function code             |
|   | INT 67H                      | ; call the memory manager        |
|   | OR AH,AH                     | ; check EMM status               |
|   | JNZ EMM_ERR_HANDLER          | ; jump to error handler on error |
|   | MOV emm_handle, DX           | ; save EMM handle                |



## INT 67H

## Map/Unmap Handle Page

## ファンクション No. 5

**機能** ハンドルページのマッピング／アンマッピング

**コール** AH = 44H  
 AL 物理ページ番号  
 BX 論理ページ番号  
 DX EMM ハンドル (ファンクション 4 で取得したハンドル)

**リターン** AH = 00H 正常実行 (ページはマッピングされた)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 83H 回復不可 (指定の EMM ハンドルがない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8AH 復旧可能 (指定の論理ページが EMM ハンドルにアロケートされている論理ページの範囲を超えた。または論理ページがアロケートされていない)  
 = 8BH 復旧可能 (物理ページ番号が、実際の物理ページの範囲を超えた。範囲内にある物理ページにマッピングし直すことにより、復旧が可能)

**解説** このファンクションは、システムメモリ内のマッピング可能な領域にある指定の物理ページに論理ページをマッピングします。物理ページ番号の最小値は実メモリ範囲外のメモリ領域と関連しています。システム内のどの物理ページがマッピング可能か、また、指定の物理番号に対応するセグメントアドレスを調べるためには、ファンクション 25 (マッピング可能な物理アドレス配列の取得) を使用します。ファンクション 25 は、物理ページ番号とセグメントアドレス間のクロスリファレンスを準備します。

このファンクションでは、関連する論理ページにFFFFHをセットすることにより物理ページをアンマッピングすることが可能です (物理ページの解除)。アンマッピングした場合は、物理ページを介して論理ページへアクセス (読み／書き) することができないので、物理ページへのアクセス (読み／書き) は行わないでください。

たとえば、プログラムを読み込んだり実行したりする前にマッピングされたページすべてを解除します。そうすることにより、読み込んだプログラムが拡張メモリにアクセスしても、その前のプログラムでマッピングしたページにアクセスすることがありません。しかし、物理ページを解除する前にマッピングコンテキストを保存しておかなければなりません。こうすることにより、後で復元を行い、アクセスすることができます。マッピングコンテキストを保存するにはファンクション 8, 15, 16 を用います。マッピングコンテキストを復元するにはファンクション 9, 15, 16 を用います。

#### ●パラメータの説明

##### AL = 物理ページ番号

論理ページをマッピングする物理ページの番号。

物理ページは相対的にゼロから番号付けされています。

##### BX = 論理ページ番号

ページフレーム内の物理ページにマッピングされる論理ページの番号です。論理ページは、相対的に 0 から番号付けされます。論理ページは 0 から [EMM ハンドルにアロケートされたページ数 - 1] の範囲をとりますが、BX の論理ページ番号に FFFFH をセットすると、AL 内に定義されている物理ページを介しての論理ページへアクセス読み書きが不可にされます。

例

|                              |                                  |
|------------------------------|----------------------------------|
| logical_page_number          | DW ?                             |
| physical_page_number         | DB ?                             |
| emm_handle                   | DW ?                             |
| MOV DX, emm_handle           | ; load EMM handle                |
| MOV BX, logical_page_number  | ; load logical page number       |
| MOV AL, physical_page_number | ; load physical page number      |
| MOV AH, 44H                  | ; load function code             |
| INT 67H                      | ; call the memory manager        |
| OR AH, AH                    | ; check EMM status               |
| JNZ EMM_ERR_HANDLER          | ; jump to error handler on error |



## INT 67H

## Deallocate Pages

## ファンクション No. 6

**機能** ページのデアロケート（解放）

**コール** AH = 45H  
DX EMM ハンドル（ファンクション 4 で取得したハンドル）

**リターン** AH = 00H 正常実行（指定の EMM ハンドルは解除された）  
 = 80H 回復不可（メモリマネージャソフトウェアが動作しない）  
 = 81H 回復不可（拡張メモリハードウェアが動作しない）  
 = 83H 回復不可（指定の EMM ハンドルがない）  
 = 84H 回復不可（メモリマネージャに渡されたファンクションが未定義）  
 = 86H 復旧可能（ページマッピングコンテキストのセーブまたはリストア（ファンクション 8 または 9）でエラーになった、指定の EMM ハンドル用のセーブエリア内にページマッピングレジスタのステートが含まれている。）

Save Page Map ファンクション（ファンクション 8）でセーブしたが、Restore Page Map（ファンクション 9）を実行していなかった（マッピングコンテキストをセーブしたら、EMM ハンドルを解放する前にリストアしておかなくてはなりません）。

**解説** このファンクションは、現在 EMM ハンドルにアロケートされている論理ページを解放するものです。アプリケーションプログラムが、あるページの解放を行うことにより、他のアプリケーションプログラムがそのページを使用できるようになります。また、ハンドルが解放された場合は、ハンドル名はすべて ASCII Null にセットされます。

**注意** アプリケーションプログラムは、MS-DOS へ返る前に必ずこのファンクションを実行しなければなりません。

もし実行しなければ、これらのページや EMM ハンドルを他のプログラムで使用することはできません。拡張メモリを使用するプログラムに致命的なエラーが発生したり強制終了した場合に、ページがアロケートされている可能性があるならば、トラップしなくてはなりません。

|   |                     |                                  |
|---|---------------------|----------------------------------|
| 例 | emm_handle          | DW ?                             |
|   | MOV DX,emm_handle   | ; load EMM handle                |
|   | MOV AH,45H          | ; load function code             |
|   | INT 67H             | ; call the memory manager        |
|   | OR AH,AH            | ; check EMM status               |
|   | JNZ EMM_ERR_HANDLER | ; jump to error handler on error |



## INT 67H

## Get Version

## ファンクション No. 7

**機 能** バージョンの取得

**コール** AH = 46H

**リターン** AH = 00H 正常実行 (バージョン番号が AL に返された)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 AL バージョン番号

**解 説** EMSインターフェイスの、バージョン番号を取得します。

●パラメータの説明

AL バージョン番号

バージョン番号は BCD 形式で返されます。上位 4 ビットがバージョン番号の整数部分を表し、下位 4 ビットが小数点以下の部分を表します。

たとえばバージョン 4.0 の場合は、次のように AL には 40 H になります。

|     |    |   |   |   |   |   |   |   |
|-----|----|---|---|---|---|---|---|---|
|     | AL |   |   |   |   |   |   |   |
| ビット | 7  | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|     | 0  | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
|     | 4  |   |   |   | . | 0 |   |   |

**例** emm\_version

DW ?

|     |                 |                                  |
|-----|-----------------|----------------------------------|
| MOV | AH,46H          | ; load function code             |
| INT | 67H             | ; call the memory manager        |
| OR  | AH,AH           | ; check EMM status               |
| JNZ | EMM_ERR_HANDLER | ; jump to error handler on error |
| MOV | emm_version, AL | ; save emm version               |

## INT 67H

## Save Page Map

## ファンクション No. 8

**機能** ページマップのセーブ

**コール** AH = 47H  
DX EMMハンドル

**リターン** AH = 00H 正常実行 (ページマップはセーブされた)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 83H 回復不可 (指定の EMM ハンドルがない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8CH 回復不可 (ページマッピングレジスタのステートをストアするセーブエリア内に空がないため、マップレジスタのステータはセーブされなかった)  
 = 8DH 状況により復旧可能 (セーブエリアには、プログラムが指定した EMM ハンドル用のページマッピングレジスタステータが既にセーブされている)

**解説** このファンクションは、ページマッピングレジスタ (ページのマッピング状態) の内容を、内部エリアにセーブするものです。通常、ソフトウェアまたはハードウェア割り込みが発生したときに、アクティブだった EMM ハンドルのメモリマッピングコンテキストをセーブするために使用されます。

拡張メモリを使用している常駐プログラムや割り込み処理ルーチン、デバイスドライバを書くときには、ハードウェアのマップ状態を保存しなければなりません。なぜならば、そのプログラムがハードウェア割り込み、ソフトウェア割り込み、MS-DOS によって呼び出されたときに、拡張メモリを使用している他のアプリケーションソフトウェアが稼動している可能性があるからです。

このファンクションは、常駐プログラムや割り込み処理ルーチン、デバイスドライバが初期化されたときに、割り当てられた EMM ハンドルを必要とします。これは、割り込まれた側のアプリケーションの EMM ハンドルではありません。

Save Page Map ファンクションは、ページフレームのうち、EMS のバージョン 3.x で定義されている 64 K バイトのページフレームのみに関するマップレジスタの状態を保存します。EMS バージョン 3.x の仕様で書かれたすべてのアプリケーションは、この 64 K バイトのものだけのマップレジスタの状態を保存することを必要とするため、多くのマッピング可能なページのすべてのマッピングコンテキストを保存することはメモリ効率を落すでしょう。EMS 3.x を超えた範囲のマップ可能なメモリを使用するアプリケーションは、マップレジスタの保存、復元のためにファンクション 15 または 16 を用いるべきです。

#### ●パラメータの説明

**DX = EMM ハンドル**

ソフトウェアまたはハードウェア割り込みをサービスする割り込みサービスルーチンに割り当てられている EMM ハンドル。割り込みサービスルーチンは、ページをマップする前に、ページマッピングハードウェアのステータスをセーブする必要があります。

| 例 | emmm_handle         | DW ?                             |
|---|---------------------|----------------------------------|
|   | MOV DX,emmm_handle  | ; load EMM handle                |
|   | MOV AH,47H          | ; load function code             |
|   | INT 67H             | ; call the memory manager        |
|   | OR AH,AH            | ; check EMM status               |
|   | JNZ EMM_ERR_HANDLER | ; jump to error handler on error |



## INT 67H

## Restore Page Map

## ファンクション No. 9

**機能** ページマップのリストア

**コール** AH = 48H  
DX EMM ハンドル

**リターン** AH = 00H 正常実行 (ページアップはリストアされた)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 83H 回復不可 (指定の EMM ハンドルがない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8EH 状況によっては復旧可能 (指定の EMM ハンドル用のセーブエリア内にページマッピングレジスタがない、プログラムがページマッピング状態の内容をセーブしていなかったため、Restore Page Map ファンクションはその内容をリストアできない)

**解説** 固有な EMM ハンドルのために、内部のセーブエリアから、ページマッピングレジスタの内容をリストアします。

拡張メモリを使用している常駐プログラムや割り込み処理ルーチン、デバイスドライバを書くときには、ハードウェアのマップ状態を保存しなければなりません。なぜならば、そのプログラムがハードウェア割り込み、ソフトウェア割り込み、MS-DOS によって呼び出されたときに、拡張メモリを使用している他のアプリケーションソフトウェアが稼動している可能性があるからです。

このファンクションは、常駐プログラムや割り込み処理ルーチン、デバイスドライバが初期化されたときに、割り当てられた EMM ハンドルを必要とします。これは、割り込まれた側のアプリケーションの EMM ハンドルではありません。

Save Page Map ファンクションは、ページフレームのうち、EMS のバージョン 3.x で定義されている 64K バイトのページフレームのみに関するマッピングレジスタの状態を保存します。EMS バージョン 3.x の仕様で書かれたすべてのアプリケーションは、この 64K バイトのものだけのマッピングレジスタの状態を保存するこ



とを必要とするため、多くのマップ可能なページのすべてのマッピングコンテキストを保存することはメモリ効率を落すでしょう。EMS 3.x ページフレームを超えた範囲のマップ可能なメモリを使用するアプリケーションは、マッピングレジスタの保存、復元のためにファンクション15または16を用いるべきです。

#### ●パラメータの説明

##### DX = EMM ハンドル

ソフトウェアまたはハードウェア割り込みをサービスする割り込みサービスルーチンに割り当てられている EMM ハンドル。割り込みサービスルーチンは、ページマッピングの状態をリストアする必要があります。

| 例 | emm_handle          | DW ?                             |
|---|---------------------|----------------------------------|
|   | MOV DX,emm_handle   | ; load EMM handle                |
|   | MOV AH,48H          | ; load function code             |
|   | INT 67H             | ; call the memory manager        |
|   | OR AH,AH            | ; check EMM status               |
|   | JNZ EMM_ERR_HANDLER | ; jump to error handler on error |

## INT 67H

**Reserved****ファンクション No. 10,11**

EMS拡張メモリ仕様の初期バージョンでは、ファンクション10はページマッピングレジスタのI/O配列を返しました。現在はこの番号は予約されているので、新しいプログラムでは使わないでください。

このファンクションを用いているプログラムは、それらをサポートしているハードウェア上では正常に動きます。しかし、EMSのバージョン4.0におけるファンクション16から30を用いているプログラムでは、ファンクション10と11を使用してはいけません。プログラムで、新しいファンクション（ファンクション16から30）とファンクション10、11を混ぜて使用すると、これらのファンクションは正常に動きません。

## INT 67H

## Get Handle Count

## ファンクション No. 12

**機能** ハンドルカウントの取得

**コール** AH = 4BH

**リターン** AH = 00H 正常実行 (ハンドルカウントは BX に返された)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 BX オープンしている EMM ハンドルの数 (OS のハンドル 0 も含む)  
 この数は 255 を超えることはない。

**解説** システム内の、オープンしている EMM ハンドル (OS のハンドル 0 を含む) の数を返します。

**例**

|                                |                                  |
|--------------------------------|----------------------------------|
| total_open_emm_handles         | DW ?                             |
| MOV AH,48H                     | ; load function code             |
| INT 67H                        | ; call the memory manager        |
| OR AH,AH                       | ; check EMM status               |
| JNZ EMM_ERR_HANDLER            | ; jump to error handler on error |
| MOV total_open_emm_handles, BX | ; save total active handle count |

## INT 67

## Get Handle Pages

## ファンクション No. 13

|             |                                                                                                                                                                                                                                                   |                                  |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------|
| <b>機 能</b>  | ハンドルページの取得                                                                                                                                                                                                                                        |                                  |
| <b>コール</b>  | AH = 4CH<br>DX EMM ハンドル                                                                                                                                                                                                                           |                                  |
| <b>リターン</b> | AH = 00H 正常実行 (指定ハンドルのページ数が BX に返された)<br>= 80H 回復不可 (メモリマネージャソフトウェアが動作しない)<br>= 81H 回復不可 (拡張メモリハードウェアが動作しない)<br>= 83H 回復不可 (指定の EMM ハンドルがない)<br>= 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)<br>BX 指定の EMM ハンドルにアロケートされている論理ページ数<br>この数は 2048 を超えることはない。 |                                  |
| <b>解 説</b>  | 指定の EMM ハンドルにアロケートされているページ数を返します。                                                                                                                                                                                                                 |                                  |
| <b>例</b>    | emm_handle                                                                                                                                                                                                                                        | DW ?                             |
|             | page_alloc_to_handle                                                                                                                                                                                                                              | DW ?                             |
|             | MOV DX,emm_handle                                                                                                                                                                                                                                 | ; load EMM handle                |
|             | MOV AH,4CH                                                                                                                                                                                                                                        | ; load function code             |
|             | INT 67H                                                                                                                                                                                                                                           | ; call the memory manager        |
|             | OR AH,AH                                                                                                                                                                                                                                          | ; check EMM status               |
|             | JNZ EMM_ERR_HANDLER                                                                                                                                                                                                                               | ; jump to error handler on error |
|             | MOV page_alloc_to_handle, BX                                                                                                                                                                                                                      | ; save handle pages              |



## INT 67H

## Get All Handle Pages

## ファンクション No. 14

**機能** 全ハンドルページの取得

**コール** AH = 4DH

ES : DI オープンされているすべての EMM ハンドルのコピーと、各ハンドルにアロケートされているページ数がストアされる配列のポインタ

```

handle_page_struct STRUC
 emm_handle DW ?
 pages_alloc_to_handle DW ?
handle_page_struct ENDS

```

**リターン** AH = 00H 正常実行 (全ハンドルページの情報が返された)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 BX オープンされている EMM ハンドルの総数 (OS のハンドル 0 を含む)  
 OS ハンドル 0 は、常にオープン状態のため、値にゼロが返されることはない。また 255 を超えることはない。

**解説** オープンされている EMM ハンドルと、各ハンドルにアロケートされているページ数の配列を返します。

●パラメータの説明

各ストラクチャのメンバは、以下の 2 つです。

• emm\_handle

最初のメンバはオープンしている EMM ハンドルの値を含むワードです。このファンクションが返すハンドル値は 10 進で 0 から 255 (0000H から 00FFH) の間にあり、ハンドルの最上位バイトは常にゼロです。

• pages\_alloc\_to\_handle

2 番目のメンバはオープンしている EMM ハンドルにアロケートされたページ数を含むワードです。

例

handle\_page

total\_open\_handles

```
MOV AX,SEG handle_page
MOV ES,AX
LEA DI, handle_page
MOV AH, 4DH
INT 67H
OR AH,AH
JNZ EMM_ERR_HANDLER
MOV total_open_handles, BX
```

handle\_page\_struct 255 DUP(?)

DW ?

```
; set handle page segment
;
; ES:DI handle page pointer
; load function code
; call the memory manager
; check EMM status
; jump to error handler on error
; save all handle page
```

## INT 67H

## Get Page Map

ファンクション No. 15  
コード 00H

**機能** ページマップの取得

**コール** AX = 4E00H

ES : DI dest\_page\_map

セグメント：オフセットの形式で示される，デスティネーション配列へのポインタ。要求された配列のサイズを決定するためには，“Get Size of Page Map Save Array” のサブファンクションを使用する。

**リターン** AH = 00H 正常実行（ページマップを取得できた）  
 = 80H 回復不可（メモリマネージャソフトウェアが動作しない）  
 = 81H 回復不可（拡張メモリハードウェアが動作しない）  
 = 84H 回復不可（メモリマネージャに渡されたファンクションが未定義）  
 = 8FH 回復不可（サブファンクションパラメータが無効）

dest\_page\_map

この配列はマッピングレジスタの状態を含む。また、プログラムが set サブファンクションをコールしたときに Get した状態に戻すために必要な追加情報も含まれる。

**解説** すべてのマップ可能なメモリ領域のために、マッピングコンテキストをセーブします。これはデスティネーションの配列へ拡張メモリのマッピングレジスタの内容をコピーすることにより、実行されます。このファンクションは、EMM ハンドルを必要としません。

|          |                      |                                  |
|----------|----------------------|----------------------------------|
| <b>例</b> | dest_page_map        | DB ? DUP (?)                     |
| MOV      | AX,SEG dest_page_map | ; set dest page map segment      |
| MOV      | ES,AX                | ;                                |
| LEA      | DI,dest_page_map     | ; ES:DI dest_page_map            |
| MOV      | AX,4E00H             | ; load function code             |
| INT      | 67H                  | ; call the memory manager        |
| OR       | AH,AH                | ; check EMM status               |
| JNZ      | EMM_ERR_HANDLER      | ; jump to error handler on error |



## INT 67H

## Set Page Map

ファンクション No. 15  
コード 01H

**機能** ページマップのセット

**コール** AX = 4E01H

DS : SI source\_page\_map

セグメント：オフセットの形式で示される，ソース配列のポインタ．アプリケーションプログラムでは，マッピング状態を含む配列をポイントしなければならない。

**リターン** AH = 00H 正常実行（ページマップはセットできた）  
 = 80H 回復不可（メモリマネージャソフトウェアが動作しない）  
 = 81H 回復不可（拡張メモリハードウェアが動作しない）  
 = 84H 回復不可（メモリマネージャに渡されたファンクションが未定義）  
 = 8FH 回復不可（サブファンクションパラメータが無効）  
 = A3H 回復不可（ソース配列の内容が不正または渡されたポインタが無効）

**解説** ページマップのサブファンクションは，システム内の各拡張メモリのマッピングレジスタにソース配列の内容をコピーすることにより，すべてのマップ可能なメモリ領域（実メモリと拡張メモリ）のためにマッピングコンテキストをリストアします。マッピングコンテキストをセーブまたはリストアする必要がある場合は，このファンクションをファンクション8と9の代用として使用することができます。また，このときに EMM ハンドルを使用する必要はありません。

|          |                         |                                  |
|----------|-------------------------|----------------------------------|
| <b>例</b> | source_page_map         | DB ? DUP (?)                     |
| MOV      | AX, SEG source_page_map | ; set source page map segment    |
| MOV      | DS, AX                  | ;                                |
| LEA      | SI, dest_page_map       | ; DS:SI source_page_map          |
| MOV      | AX, 4E01H               | ; load function code             |
| INT      | 67H                     | ; call the memory manager        |
| OR       | AH, AH                  | ; check EMM status               |
| JNZ      | EMM_ERR_HANDLER         | ; jump to error handler on error |



## INT 67H

## Get &amp; Set Page Map

ファンクション No. 15  
コード 02H

**機能** ページマップの取得とセット

**コール** AX = 4E02H

ES : DI dest\_page\_map

セグメント：オフセットの形式で示される、デスティネーション配列へのポインタ。現在のマップレジスタの内容は、この配列中にセーブされる。

DS : SI source\_page\_map

セグメント：オフセットの形式で示される、ソース配列へのポインタ。この配列の内容は、マップレジスタ内にコピーされる。

**リターン** AH = 00H 正常実行（ページマップの Get & Set ができた）  
 = 80H 回復不可（メモリマネージャソフトウェアが動作しない）  
 = 81H 回復不可（拡張メモリハードウェアが動作しない）  
 = 84H 回復不可（メモリマネージャに渡されたファンクションが未定義）  
 = 8FH 回復不可（サブファンクションパラメータが無効）  
 = A3H 回復不可（ソース配列の内容が不正または渡されたポインタが無効）

dest\_page\_map

この配列は、マッピングレジスタのステートを含む。プログラムが Set サブファンクションをコールしたときに Get した状態に戻すために必要な追加情報も含まれる。

**解説** このサブファンクションは、まず、システム内の各拡張メモリの、マッピングレジスタの内容をデスティネーション配列に、コピーします。次にサブファンクションはソース配列の内容を、各拡張メモリのマッピングレジスタにコピーします。これにより、カレントのマッピングコンテキストをセーブし、すべてのマップ可能なメモリ領域（内部、拡張メモリ共）のために、前のマッピングコンテキストをリストアします。

|     |                        |                                  |
|-----|------------------------|----------------------------------|
| 例   | dest_page_map          | DB ? DUP (?)                     |
|     | source_page_map        | DB ? DUP (?)                     |
| MOV | AX,SEG dest_page_map   | ; set dest page map segment      |
| MOV | ES,AX                  | ;                                |
| MOV | AX,SEG source_page_map | ; set source page map segment    |
| MOV | DS,AX                  | ;                                |
| LEA | DI,dest_page_map       | ; ES:DI dest_page_map            |
| LEA | SI,dest_page_map       | ; DS:SI source_page_map          |
| MOV | AX,4E02H               | ; load function code             |
| INT | 67H                    | ; call the memory manager        |
| OR  | AH,AH                  | ; check EMM status               |
| JNZ | EMM_ERR_HANDLER        | ; jump to error handler on error |

## INT 67H

## Get Size of Page Map Save Array

ファンクション No. 15  
コード 03H

**機能** ページマップセーブ配列のサイズ取得

**コール** AX = 4E03H

**リターン** AH = 00H 正常実行 (配列のサイズを取得した)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (サブファンクションパラメータが無効)

AL size\_of\_array

プログラムが前項の3つのサブファンクションをコールするときに必要とされるメモリエリアのバイト数。このメモリエリアは、アプリケーションプログラムで供給する。

**解説** 前項に記した3つのサブファンクションにより渡された配列に必要なメモリの大きさを返します。ページマップのセーブ配列は、拡張メモリシステムの構成やマネージャの動作に依存しています。そのため、サイズはメモリマネージャがロードされた後に取得されなければなりません。

|          |                  |                                  |
|----------|------------------|----------------------------------|
| <b>例</b> | size_of_array    | DB ?                             |
| MOV      | AX,4E03H         | ; load function code             |
| INT      | 67H              | ; call the memory manager        |
| OR       | AH,AH            | ; check EMM status               |
| JNZ      | EMM_ERR_HANDLER  | ; jump to error handler on error |
| MOV      | size_of_array,AL | ; save size of array             |



## INT 67H

## Get Partial Page Map

ファンクション No. 16  
コード 00H

**機能** システム内の指定のマップ可能メモリ領域用のマッピングコンテキストの一部をセーブする。

**コール** AX = 4F00H

DS : SI partial\_page\_map

ページマップの一部を示すストラクチャのポインタ。

```
partial_page_map_struct STRUC
 mappable_segment_count DW ?
 mappable_segment DW (?) DUP (?)
partial_page_map_struct ENDS
```

ES : DI dest\_array

要求された配列の大きさを決定するためには、Get Size of Partial map Save Array を使用する。

**リターン** AH = 00H 正常実行 (ページマップの取得ができた)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8BH 回復不可 (指定のセグメントのうち、マップ不可能なセグメントがある)  
 = 8FH 回復不可 (サブファンクションパラメータが無効)  
 = A3H 回復不可 (ソース配列の内容が不正または渡されたポインタが無効)

dest\_array

この配列は、マッピングコンテキストの一部と、プログラムが Set ファンクションをコールしたとき、このコンテキストを元の状態へリストアするために必要な追加情報などが含まれる。



**解 説**

このファンクションはシステム内で指定されたマップ可能メモリ領域のマッピングコンテキストの一部をセーブします。また、このファンクションではマッピングコンテキストの一部のみをセーブするため、ファンクション 15 に比べて、セーブエリア用に確保するメモリもかなり削減され、同時に処理速度も速くなります。このファンクションは、これらの処理を、デスティネーション配列に選択されたマッピングコンテキストの内容をコピーすることにより行います。

## ●パラメータの説明

ストラクチャのメンバは以下の 2 つです。

## . mappable\_segment\_count

このメンバはワードで、すぐあとに続くワード配列内のメンバ数を定義します。この数はマップできるセグメントの数を超えてはなりません。

## . mappable\_segment

2 番目のメンバは、ワードでセーブされるマッピングコンテキストを所有するマップ可能なメモリ領域のセグメントアドレスを含みます。このセグメントアドレスは、マップ可能なセグメントでなければなりません。どのセグメントがマップ可能かを調べるには、ファンクション 25 を使用します。

**例**

|                             |                                  |
|-----------------------------|----------------------------------|
| partial_page_map            | partial_page_map_struct < >      |
| dest_array                  | DB ? DUP (?)                     |
| MOV AX,SEG partial_page_map | ; partial page map segment       |
| MOV DS,AX                   | ;                                |
| LEA SI,partial_page_map     | ; DS : SI partial page map point |
| MOV AX,SEG dest_array       | ; set dest array segment         |
| MOV ES,AX                   | ;                                |
| LEA DI,dest_array           | ; ES : DI dest array point       |
| MOV AX,4F00H                | ; load function code             |
| INT 67H                     | ; call the memory manager        |
| OR AH,AH                    | ; check EMM status               |
| JNZ EMM_ERR_HANDLER         | ; jump to error handler on error |

## INT 67H

## Set Partial Page Map

ファンクション No. 16  
コード 01H

**機能** メモリ内のマッピングコンテキストの一部をリストアする。

**コール** AX = 4F01H

DS : SI source\_array

セグメント：オフセットの形式を持つソース配列のポインタ。アプリケーションプログラムでは、マッピングレジスタステートの一部を含む配列をポイントしなければならない。要求された配列のサイズを決定するためには、Get Size of Partial Page Map Save Arrayファンクションを参照すること。

**リターン** AH = 00H 正常実行（ページマップのセットができた）  
 = 80H 回復不可（メモリマネージャソフトウェアが動作しない）  
 = 81H 回復不可（拡張メモリハードウェアが動作しない）  
 = 84H 回復不可（メモリマネージャに渡されたファンクションが未定義）  
 = 8FH 回復不可（サブファンクションパラメータが無効）  
 = A3H 回復不可（ソースアレイの内容が不正または渡されたポインタが無効）

**解説** このファンクションは、指定されたマップ可能領域のために、マッピングコンテキストの一部をリストアするものです。このファンクションでは、マッピングコンテキストの一部のみをリストアするため、全システムのマッピングコンテキストをリストアするファンクション 15 に比べて、メモリエリアは削減され、また処理速度も速くなります。サブファンクションは、ソース配列の内容を選択されたマッピングコンテキストにコピーすることによりセットを行います。

**例**

|     |                     |                                  |
|-----|---------------------|----------------------------------|
| MOV | AX,SEG source_array | ; source array segment           |
| MOV | DS,AX               | ;                                |
| LEA | SI,source_array     | ; DS:SI source array point       |
| MOV | AX,4F01H            | ; load function code             |
| INT | 67H                 | ; call the memory manager        |
| OR  | AH,AH               | ; check EMM status               |
| JNZ | EMM_ERR_HANDLE      | ; jump to error handler on error |



## INT 67H

## Get Size of Partial Page Map Save Array

ファンクション No. 16  
コード 02H

**機能** ページマップの一部をセーブする配列のサイズ取得

**コール** AX = 4F02H

BX 部分的にマップされるページ数

この数は、Get Partial Page Mapサブファンクション中のmappable\_segment countと同じになる。

**リターン** AH = 00H 正常実行 (配列のサイズを取得した)

= 80H 回復不可 (メモリマネージャソフトウェアが動作しない)

= 81H 回復不可 (拡張メモリハードウェアが動作しない)

= 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)

= 8BH 回復不可 (物理ページ数がシステム内の物理ページの範囲を超えた)

= 8FH 回復不可 (サブファンクションパラメータが無効)

AL size\_of\_partial\_save\_array

プログラムが、“Get” “Set” のサブファンクションをコールするときに、必要とされるバイト数。このエリアはアプリケーションプログラム側で用意する。

**解説**

ファンクション 16 の他の 2 つのファンクションにより渡された配列に必要なメモリの大きさを返します。ページマップのセーブ配列のサイズは、拡張メモリの構成やマネージャの動作に依存しています、そのためハードウェアの構成や動作間に関係があるため、指定のメモリマネージャがロードされた後にサイズを調べなくてはなりません。

**例**

number\_of\_page\_to\_map DW ?

size\_of\_partial\_save\_array DW ?

MOV BX,number\_of\_page\_to\_map ; set number of page to map

MOV AX,4F02H ; load function code

INT 67H ; call the memory manager

OR AH,AH ; check EMM status

JNZ enn\_err\_hundler ; jump to error handler on error

MOV size\_of\_partial\_save\_array, AX ; save size of partial save array

## INT 67H

|                                                                                                                    |
|--------------------------------------------------------------------------------------------------------------------|
| <b>Map/Unmap Multiple Handle Pages</b> <b>ファンクション No. 17</b><br>(Logical Page/Physical Page Method) <b>コード 00H</b> |
|--------------------------------------------------------------------------------------------------------------------|

**機能**    複数ハンドルページのマッピング／物理ページのアンマッピング  
 (論理ページ／物理ページ方式)

**コール**    AX = 5000H  
 DX EMM ハンドル  
 CX 配列内のエントリ数  
 DS : SI 配列のストラクチャへのポインタ

```

log_to_phys_map_struct STRUC
 log_page_number DW ?
 phys_page_number DW ?
log_to_phys_map_struct ENDS

```

**リターン**    AH = 00H 正常実行 (マッピング完了)  
               = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
               = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
               = 83H 回復不可 (指定の EMM ハンドルがない)  
               = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
               = 8AH 復旧可能 (マッピングされた論理ページが EMM ハンドルにアロケートされている論理ページの範囲を超えている)  
               = 8BH 復旧可能 (物理ページが マッピング可能な物理ページの範囲を超えている)  
               = 8FH 回復不可 (サブファンクションパラメータが無効)

**解説**    指定された複数の論理ページを、指定の物理ページにマッピングします。または指定の物理ページをアンマッピングします。

#### パラメータの説明

ストラクチャのメンバは以下の2つです。

#### .log\_page\_number

このメンバはワードで、マッピングされる論理ページ番号を示します。論理ページの番号は、0から[EMMハンドルにアロケートされている論理ページの最大数-1]までの範囲にあります。もし論理ページがFFFFHにセットされていれば、指



定の物理ページはアンマッピングされ、物理ページを介して論理ページへアクセス（読み書き）できなくなります。

#### phys\_page\_number

2 番目のメンバはワードで、論理ページがマッピングされる物理ページの番号を示します。物理ページの番号は 0 から [システムがサポートしている物理ページの最大数-1] までの範囲にあります。

例

```

log_to_phys_map log_to_phys_map_struct ? DUP (?)
emm_handle DW ?

MOV AX,SEG log_to_phys_map ; set log to phys map segment
MOV DS,AX ;
LEA SI,log_to_phys_map ; DS:SI log to phys map point
MOV CX,LENGTH log_to_phys_map ; set array entry
MOV DX,emm_handle ; set handle
MOV AX,5000H ; load function code
INT 67H ; call the memory manager
OR AH,AH ; check EMM status
JNZ emm_err_handler ; jump to error handler on error

```

## INT 67H

|                                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------|
| <b>Map/Unmap Multiple Handle Pages</b> <b>ファンクション No. 17</b><br>(Logical Page/Segment Address Method) <b>コード 01H</b> |
|----------------------------------------------------------------------------------------------------------------------|

**機 能**    複数ハンドルページのマッピング／物理ページのアンマッピング  
 (論理ページ／セグメントアドレス方式)

**コール**    AX = 5001H  
 DX   EMM ハンドル  
 CX   配列内のエントリ数  
 DS : SI   配列のストラクチャへのポインタ

```

log_to_seg_map_struct STRUC
 log_page_number DW ?
 mappable_segment_address DW ?
log_to_seg_map_struct ENDS

```

**リターン**    AH = 00H   正常実行 (マッピング完了)  
               = 80H   回復不可 (メモリマネージャソフトウェアが動作しない)  
               = 81H   回復不可 (拡張メモリハードウェアが動作しない)  
               = 83H   回復不可 (指定の EMM ハンドルがない)  
               = 84H   回復不可 (メモリマネージャに渡されたファンクションが未定義)  
               = 8AH   復旧可能 (マッピングされた論理ページが EMM ハンドルにアロケ  
                               ートされている論理ページの範囲を超えている)  
               = 8BH   復旧可能 (指定されたマップ可能なセグメントアドレスがマップ  
                               ングできない)  
               = 8FH   回復不可 (サブファンクションパラメータが無効)

**解 説**    指定された複数の論理ページを指定のセグメントアドレスにマッピングしま  
 す。または指定の物理ページをアンマッピングします。

●パラメータの説明

ストラクチャのメンバは以下の2つです。

. log\_page\_number

このメンバはワードで、マッピングされる論理ページの番号を示します。論理ペ  
 ージの番号は0から [アロケートされる論理ページの最大数-1] の範囲にありま  
 す。もし論理ページ番号がFFFFHにセットされていれば、指定の物理ページはア

ンマッピングされ、物理ページを介して論理ページへアクセス（読み書き）できなくなります。

#### . mappable\_segment\_addres

2番目のメンバはワードで、論理ページがマッピングされるセグメントアドレスを示します。このセグメントアドレスは、マッピング可能なセグメントアドレスでなくてはなりません。マッピング可能なセグメントアドレスは、ファンクション 25 (Get Mappable Physical Address Array) により知ることができます。

#### 例

```
log_to_phys_map log_to_phys_map_struct ? DUP (?)
emm_handle DW ?

MOV AX,SEG log_to_phys_map ; set log to phys map segment
MOV DS,AX ;
LEA SI,log_to_phys_map ; DS:SI log to phys map point
MOV CX,LENGTH log_to_phys_map ; set array entry
MOV DX,emm_handle ; set handle
MOV AX,5001H ; load function code
INT 67H ; call the memory manager
OR AH,AH ; check EMM status
JNZ emm_err_handler ; jump to error handler on error
```

#### 参考

このファンクション（ファンクション 17 の 2 つのファンクション）1 回のコールで、システムがサポートしている物理ページと同数の論理ページをマッピング（アンマッピング）できます。したがって、1 度に 1 つのページをマッピングするより少ない処理時間で済みます。多くのページをマッピングするアプリケーションプログラムでは、このファンクションがより良いマッピング方法です。

#### ●複数ページのマッピング

このファンクションに渡された EMM ハンドルは、どのタイプの論理ページがマッピングされたかを決定します。ファンクション 4 やファンクション 27 (Allocate Standard Pages サブファンクション) でアロケートされた論理ページは、ページとして参照され、そのサイズは 16K バイトです。ファンクション 27 (Allocate Raw Pages サブファンクション) でアロケートされた論理ページはローページとして参照されますが、ファンクション 4 でアロケートされたページと同サイズとは限りません。



### ●複数ページのアンマッピング

このファンクションは、指定の物理ページを介した論理ページの読み書きを不可にすることができます。指定の物理ページからアンマッピングされた論理ページは、その物理ページから読み書きすることはできません。(アンマッピングされた) 論理ページは、再度同じ場所にマッピングするかまたはアンマッピング状態の物理ページにマッピングすることで、再び使用可能にします。物理ページのアンマッピングは論理ページをFFFFHにセットすることにより完了します。

### ●複数ページのマッピングとアンマッピングの並行実行

ページのマッピングとアンマッピングは一度のコールで実行することができます。

マッピングまたはアンマッピング対象のページが存在しなくても、エラー扱いにはなりません。もし、ゼロページのマッピングまたはアンマッピングの要求が実行されても、何も行われなし、また何のエラーも返しません。

### ●マッピングとアンマッピング方法の変更

ページのマッピング、アンマッピングには2つの方法があります。どちらの方法でも結果は同じです。

1. 第一の方法は、論理ページとそれがマッピングされる物理ページを指定します。この方法はファンクション5 (Map Unmap Handle Page) の拡張です。
2. 第2の方法は、論理ページとそれがマッピングされるセグメントアドレスを指定します。

これは、基本的には第1の方法と同じなのですが、ページの相対位置を表しているだけの番号を使用するよりも、物理ページの実際のアドレスを使用した方がより簡単です。メモリマネージャは、指定のセグメントアドレスが、マッピング可能な物理ページの範囲内にあるかどうかをチェックします。そしてマネージャは、渡されたセグメントアドレスを、ページにマッピングするために必要な内部表現に置き換えます。

## INT 67H

## Reallocate Pages

## ファンクション No. 18

**機 能** ページの再アロケート

**コール** AH = 51H  
DX EMM ハンドル  
BX reallocation\_count

このファンクションがコールされた後で、このハンドルがアロケートするページの総数

**リターン** AH = 00H 正常実行 (再アロケート完了)  
= 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
= 81H 回復不可 (拡張メモリハードウェアが動作しない)  
= 83H 回復不可 (指定の EMM ハンドルがない)  
= 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
= 87H 復旧可能 (システム内の使用可能なページ数は新しい再アロケーション要求にとって無意味であった。プログラムは、より少ないページを EMM にアロケートするよう指定することにより復旧できる)  
= 88H 復旧可能 (未アロケートページの数新しいアロケーション要求にとって無意味であった。プログラムは、追加ページが使用可能なときに再度要求をだすか、より少ないページを指定することにより復旧できる)

BX 再アロケーション後ハンドルにアロケートされたページ数

**解 説** このファンクションにより、EMM ハンドルにアロケートされている論理ページの数を増したり減らしたりできます。再アロケートには以下の 4 つの場合があります。

●パラメータの説明

BX reallocation\_count

1. 再アロケーションカウント = 0

アプリケーションに割り当てられたハンドルは割り当てられたままで、まだアプリケーションがこれを使用することができます。メモリマネージャは、ハンドルを別のアプリケーションに再び割り当てることはしません。しかしハンドルは、



メモリマネージャに返した、アロケートされていたすべてのページをも保持しています。アプリケーションは、DOSに戻る前に Deallocate Pages ファンクション (ファンクション 6) を呼び出さなければなりません。さもなければ、ハンドルは割り当てられたままであり、別のアプリケーションが使用することはできません。

## 2. 再アロケーションカウント = カレントのアロケーションカウント

これはエラーとしては扱いません。成功ステータス (AH=0) を返します。

## 3. 再アロケーションカウント > カレントのアロケーションカウント

メモリマネージャは、指定された EMM ハンドルに既にアロケートされているページに、新たなページを増やそうとします。加えられた新たなページ数は、再アロケート数と現在のアロケート数の差です。EMM ハンドルにアロケートされていた論理ページの順番は、この操作の後も変わりません。新たにアロケートされたページは、前にアロケートされたページが終ったところから昇順に始まる論理ページ番号が付けられます。

## 4. 再アロケーションカウント < カレントのアロケーションカウント

メモリマネージャは、現在アロケートされたページのいくつかを取り除き、それらをメモリマネージャに返そうとします。取り除かれる古いページの数、現在のアロケート数と再アロケート数の差です。ページは、指定された EMM ハンドルに現在アロケートされているページ列の最後から取り除かれます。EMM ハンドルにアロケートされた論理ページの順番は、この操作の後も変わりません。

どのような型の論理ページが再アロケートされるかは EMM ハンドルで決まります。ファンクション 4 でアロケートされた論理ページはページと呼ばれ、16K バイトの大きさを持ちます。ファンクション 27 でアロケートされた論理ページはロー (raw) ページと呼ばれ、ファンクション 4 でアロケートしたページの大きさと同じとは限りません。

### ●リターン値の説明

#### BX 再アロケーション後ハンドルにアロケートされたページ数

ページが追加または削除された後、EMM ハンドルに現在アロケートされているページ数を示します。AH に 0 が返ってきた場合は、BX の値は、このファンクションの呼び出し前のハンドルにアロケートされたページ数と等しくなります。この情報は要求どおりの結果が得られたかどうかを調べる手掛かりとなります。



|   |                                 |                                  |  |
|---|---------------------------------|----------------------------------|--|
| 例 | emm_handle                      | DW ?                             |  |
|   | realloc_count                   | DW ?                             |  |
|   | current_alloc_page_count        | DW ?                             |  |
|   | MOV DX,emm_handle               | ; set emm handle                 |  |
|   | MOV BX,realloc_count            | ; set realloc count              |  |
|   | MOV AH,51H                      | ; load function code             |  |
|   | INT 67H                         | ; call the memory manager        |  |
|   | OR AH,AH                        | ; check EMM status               |  |
|   | JNZ emm_err_handler             | ; jump to error handler on error |  |
|   | MOV current_alloc_page_count,BX | ; save current alloc page count  |  |

## INT 67H

## Get Handle Attribute

ファンクション No. 19  
コード 00H

**機能** ハンドルアトリビュートの取得

**コール** AX = 5200H  
DX EMM ハンドル

**リターン** AH = 00H 正常実行 (EMMハンドルのアトリビュートを取得した)  
= 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
= 81H 回復不可 (拡張メモリハードウェアが動作しない)  
= 83H 回復不可 (指定の EMM ハンドルがない)  
= 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
= 8FH 回復不可 (サブファンクションパラメータが無効)  
= 91H 回復不可 (サポートされていない)

AL handle\_attribute

EMM ハンドルのアトリビュートを示す。

0 の場合 ハンドルは揮発性。

1 の場合 ハンドルは不揮発性。

**解説** このファンクションは、ハンドルに関するアトリビュートを返します。アトリビュートは揮発性か不揮発性のどちらかです。不揮発性のアトリビュートのハンドルは、メモリマネージャがウォームブートの間ハンドルのページの内容をセーブできるようにします。しかし、このファンクションは、ユーザーオプションで使えない場合や、メモリボードやシステムハードウェアでサポートされていない場合があります。

ハンドルのアトリビュートが不揮発性にセットされているならば、ハンドルとその名前(割り当てられている場合)、ハンドルにアロケートされたページの内容は、ウォームブート後も全部保持されています。

**注意** PC-9800シリーズでは、揮発性のアトリビュートのみ通知されます。

例

emm\_handle

DW ?

handle\_attribute

DB ?

MOV DX,emm\_handle

; set emm handle

MOV AX,5200H

; load function code

INT 67H

; call the memory manager

OR AH,AH

; check EMM status

JNZ emm\_err\_handler

; jump to error handler on error

MOV handle\_attribute,AL

; save handle attribute



## INT 67H

## Set Handle Attribute

ファンクション No. 19  
コード 01H

**機能** ハンドルアトリビュートのセット

**コール** AX = 5201H

DX EMM ハンドル

BL new\_handle\_attribute

EMM ハンドルの新しいアトリビュートを示す

**リターン** AH = 00H 正常実行 (EMM ハンドルのアトリビュートをセットした)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 83H 回復不可 (指定の EMM ハンドルがない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (サブファンクションパラメータが無効)  
 = 90H 回復不可 (指定のアトリビュートは定義されていない)  
 = 91H 回復不可 (サポートされていない)

**解説** このファンクションは、EMM ハンドルに関するアトリビュートを修正するときなどに使用します。EMM ハンドルが持つアトリビュートは、揮発性または不揮発性です。不揮発性アトリビュートは、EMM がウォームブートの間ハンドルのページの内容を保持できるようにします。しかし、このファンクションはユーザーオプションで使用できない場合や、メモリボードやシステムハードウェアでサポートされていない場合があります。もし EMM ハンドルのアトリビュートが不揮発性にセットされていれば、EMM ハンドル、または、EMM ハンドルの名前(割り当てられている場合)、EMM ハンドルにアロケートされているページの内容は、ウォームブート後も全部保持されます。

●パラメータの説明

BL EMM ハンドルの新しいアトリビュート

0 の場合 EMM ハンドルは揮発性

1 の場合 EMM ハンドルは不揮発性

揮発性の EMM ハンドルアトリビュートは、メモリマネージャに、ウォームブートした後 EMM ハンドルにアロケートされているページと EMM ハンドルの

両方をデアロケートするように知らせます。もし、全 EMM ハンドルが揮発性のアトリビュート(既定のアトリビュート)ならば、EMM ハンドルのディレクトリは空になり、拡張メモリすべてがウォームブート後すぐに 0 に初期化されます。

**注 意** PC-9800 シリーズでは、不揮発性のアトリビュートをサポートしていません。不揮発性を指定した場合には、AH=91H が返されます。

**例**

|                   |                      |   |                                  |
|-------------------|----------------------|---|----------------------------------|
| emm_handle        | DW                   | ? |                                  |
| new_handle_attrib | DB                   | ? |                                  |
| MOV               | DX,emm_handle        |   | ; set emm handle                 |
| MOV               | BX,new_handle_attrib |   | ; set new handle attribute       |
| MOV               | AX,5201H             |   | ; load function code             |
| INT               | 67H                  |   | ; call the memory manager        |
| OR                | AH,AH                |   | ; check EMM status               |
| JNZ               | emm_err_handler      |   | ; jump to error handler on error |

## INT 67H

## Get Attribute Capability

ファンクション No. 19  
コード 02H

**機能** ハンドルアトリビュートのケイパビリティを取得

**コール** AX = 5202H

**リターン** AH = 00H 正常実行 (EMM ハンドルのアトリビュートを取得した)  
 = 80H 回復不可 (メモリアネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 84H 回復不可 (メモリアネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (サブファンクションパラメータが無効)

AL attribute\_capability

アトリビュートの性能を示す。

0 の場合、メモリアネージャとハードウェアは揮発性の EMM ハンドルのみをサポート。

1 の場合、メモリアネージャとハードウェアは不揮発性の EMM ハンドルと揮発性の EMM ハンドルの両方をサポート。

**解説** このファンクションは、メモリアネージャが不揮発性のアトリビュートをサポートできるかを調べるために使用します。

**注意** PC-9800 シリーズでは、常に揮発性のアトリビュート (AL=0) を返します。

|          |                      |                                  |
|----------|----------------------|----------------------------------|
| <b>例</b> | attrib_capability    | DB ?                             |
| MOV      | AX,5202H             | ; load function code             |
| INT      | 67H                  | ; call the memory manager        |
| OR       | AH,AH                | ; check EMM status               |
| JNZ      | emm_err_handler      | ; jump to error handler on error |
| MOV      | attrib_capability,AL | ; save attribute capability      |



## INT 67H

## Get Handle Name

ファンクション No. 20  
コード 00H

**機能** ハンドル名の取得

**コール** AX = 5300H  
DX EMM のハンドル番号  
ES : DI handle\_name 配列

8 バイトの配列で、現在 EMM ハンドルに割り当てられている名前がコピーされる。

**リターン** AH = 00H 正常実行 (ハンドル名を取得した)  
= 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
= 81H 回復不可 (拡張メモリハードウェアが動作しない)  
= 83H 回復不可 (指定の EMM ハンドルがない)  
= 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
= 8FH 回復不可 (サブファンクションパラメータが無効)

handle\_name 配列

指定の EMM ハンドルの名前を含む

**解説** このファンクションは、現在 EMM ハンドルに割り当てられている 8 文字の名前を取得します。ハンドル名に使用されるキャラクタについては制限はありません (00H~FFH のいずれのコードでもよい)。ハンドル名は、ASCII の null (バイナリの 0) に 3 回初期化されます。それは、メモリマネージャがインストールされたとき、ハンドルがアロケートされるとき、EMM ハンドルがデアロケートされるときの 3 回です。ハンドル名がすべて ASCII の null ならば、名前はないと見なされます。EMM ハンドルが名前を割り当てられるとき、少なくとも名前中の 1 文字は、名前のない EMM ハンドルと区別するために null でないキャラクタでなくてはなりません。

|          |                         |                             |
|----------|-------------------------|-----------------------------|
| <b>例</b> | handle_name             | DB 8 DUP (?)                |
|          | emm_handle              | DW ?                        |
|          | MOV AX, SEG handle_name | ; set handle name segment   |
|          | MOV ES, AX              | ;                           |
|          | LEA DI, handle_name     | ; ES:DI handle name pointer |

```

MOV DX,emm_handle ; set emm handle
MOV AX,5300H ; load function code
INT 67H ; call the memory manager
OR AH,AH ; check EMM status
JNZ emm_err_handler ; jump to error handler on error

```

## INT 67H

## Set Handle Name

ファンクション No. 20  
コード 01H

**機能** ハンドル名のセット

**コール** AX = 5301H  
DX EMM のハンドル番号  
DS : SI handle\_name へのポインタ

EMM ハンドルに割り当てられる名前を含むバイトの配列、ハンドル名が 8 バイトに満たない場合は残りを null で埋める。

**リターン** AH = 00H 正常実行 (ハンドル名をセットした)  
= 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
= 81H 回復不可 (拡張メモリハードウェアが動作しない)  
= 83H 回復不可 (指定の EMM ハンドルがない)  
= 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
= 8FH 回復不可 (サブファンクションパラメータが無効)  
= A1H 復旧可能 (この名前を持つハンドルは既に存在する、指定のハンドルに名前の割り当てができない。)

**解説** このファンクションは、EMM ハンドルに対し 8 文字の名前を割り当てます。ハンドル名で使われるキャラクタに制限はありません。全文字 (ASCII コード 0 ~ FFH) を名前の各キャラクタとして割り当てられます。

インストール時、すべての EMM ハンドルには ASCII の null (バイナリの 0) で初期化された名前がつけられています。名前の中が全部 ASCII の null ならば、名前はありません、EMM ハンドルが名前を割り当てられたとき、少なくとも 1 文字は、名前のない EMM ハンドルと区別するために null ではない ASCII キャラクタでなければならず、また同じ名前を他の EMM ハンドルが持つことはできません。

EMM ハンドルは、EMM ハンドルに新しい値をセットすることで名前を付け直すことが可能です。EMM ハンドルは、ハンドル名を全部 ASCII の null にセットすることにより名前を除去することも可能です。EMM ハンドルがデアロケートされると、名前はなくなります。



例

|                        |                                  |
|------------------------|----------------------------------|
| handle_name            | DB "AARDVARK"                    |
| emm_handle             | DW ?                             |
| MOV AX,SEG handle_name | ; set handle name segment        |
| MOV DS,AX              | ;                                |
| LEA SI,handle_name     | ; DS:SI handle name pointer      |
| MOV DX,emm_handle      | ; set emm handle                 |
| MOV AX,5301H           | ; load function code             |
| INT 67H                | ; call the memory manager        |
| OR AH,AH               | ; check EMM status               |
| JNZ emm_err_handler    | ; jump to error handler on error |

## INT 67H

## Get Handle Directory

ファンクション No. 21  
コード 00H

**機能** ハンドルのディレクトリ取得

**コール** AX = 5400H

ES : DI handle\_dir へのポインタ

メモリマネージャがハンドルのディレクトリをコピーするメモリエリアへのポインタ

```

Handle_dir_struct STRUC
 handle_value DW ?
 handle_name DB 8 DUP(?)
Handle_dir_struct ENDS

```

**リターン** AH = 00H 正常実行 (メモリマネージャありハードウェアは正常に動作)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 83H 回復不可 (指定の EMM ハンドルがない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (サブファンクションパラメータが無効)

handle\_dir

EMM ハンドルの値と各 EMM ハンドルに関連するハンドル名を含む。

AL handle\_dir 配列内のエントリ数

ハンドルのディレクトリ配列内のエントリ数を示す。これはオープンな EMM ハンドルの数とも同じ。たとえば、オープンな EMM ハンドルが 1 つであれば AL には 1 が返される。

**解説** このファンクションは、すべてのアクティブ EMM ハンドルと各ハンドルに割り当てられている名前を返します。名前を割り当てられない EMM ハンドルは、ASCII の null (バイナリで 0) で埋まったデフォルト名を持ちます。EMM ハンドルが最初にアロケートされたとき、または EMM ハンドルに属するすべてのページがデアロケートされたとき (つまりオープン EMM ハンドルがクローズされたとき)、EMM ハンドルのデフォルト名は ASCII の null にセットされます。これは、後で EMM ハンドルがオープンされたとき、名前を持てるようにするためで

す。名前に割り当てられる値は 0~FFH です。

配列に要求されるバイト数は 10 バイト×EMM ハンドルの合計数(1 エントリ  
当り 10 バイト) でこの配列の大きさの上限は 10 バイト×255=2550 バイトです。

#### ●パラメータの説明

ストラクチャのメンバは以下の 2 つです。

**handle\_value**

最初のメンバはワードで、オープンな EMM ハンドルを示します。

**handle\_name**

2 番目のメンバは EMM ハンドルの ASCII 名を含む 8 バイトの配列です。現在  
ハンドルに名前がなければ、すべてゼロの値 (ASCII の null) がセットされて  
います。

例

```
handle_dir handle_dir_struct 255 DUP (?)
num_entries_in_handle_dir DB ?
```

```
MOV AX,SEG handle_dir ; set handle dir segment
MOV ES,AX ;
LEA DI,handle_dir ; ES:DI handle dir pointer
MOV AX,5400H ; load function code
INT 67H ; call the memory manager
OR AH,AH ; check EMM status
JNZ emm_err_handler ; jump to error handler on error
MOV num_entries_in_handle_dir,AL
 ; save number of entries in handle dir
```



## INT 67H

## Search for Named Handle

ファンクション No. 21  
コード 01H**機能** 指定の名前を持つ EMM ハンドルのサーチ**コール** AX = 5401H

DS : SI handle\_name

サーチする名前を含む 8 バイトの文字列へのポインタ

**リターン** AH = 00H 正常実行 (指定の名前のついた EMM ハンドルが見つかった)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (サブファンクションパラメータが無効)  
 = A0H 復旧可能 (EMM ハンドルが見つからない)  
 = A1H 回復不可 (ハンドル名がない、全部 null コード)

DX 指定した名前に一致したハンドルの値

**解説** このファンクションは、固有の名前を持つハンドルを、ハンドル名のディレクトリからサーチします。もし、その名前のついたハンドルが見つければ、このファンクションは、その名前のハンドル番号を返します。すべてのハンドルは、インストール時にその名前はすべて ASCII の null にセットされます。名前のすべてが ASCII の null であるハンドルは、名前を持っていないものとみなします。ハンドルが名前を割り当てるとき、名前のないハンドルと区別するために、少なくとも 1 文字は null 以外のキャラクタでなければなりません。

**例** named\_handle DB 'AARDVARK'  
 named\_handle\_value DW ?

```

MOV AX,SEG named_handle ; set named handle segment
MOV DS,AX ;
LEA SI,named_handle ; DS:SI named handle pointer
MOV AX,5401H ; load function code
INT 67H ; call the memory manager
OR AH,AH ; check EMM status
JNZ emm_err_handler ; jump to error handler on error
MOV named_handle_value,DX ; save named handle value

```

## INT 67H

## Get Total Handles

ファンクション No. 21  
コード 02H

**機能** ハンドルの総数の取得

**コール** AX = 5402H

**リターン** AH = 00H 正常実行 (サポートされているハンドルの総数を返した)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (サブファンクションパラメータが無効)

BX total\_handles

この値は、プログラムがメモリマネージャにアロケート要求の出せるハンドルの最大数を示す。この値は OS ハンドル (ハンドル値 0) も含む。

**解説** このファンクションは、オペレーティングシステムの EMM ハンドル (ハンドル値 0) を含め、メモリマネージャがサポートしている EMM ハンドルの総数を返します。

|          |                  |                                  |
|----------|------------------|----------------------------------|
| <b>例</b> | total_handles    | DW ?                             |
| MOV      | AX,5402H         | ; load function code             |
| INT      | 67H              | ; call the memory manager        |
| OR       | AH,AH            | ; check EMM status               |
| JNZ      | emm_err_handler  | ; jump to error handler on error |
| MOV      | total_handles,BX | ; save total handle              |

## INT 67H

## Alter Page Map &amp; Jump

## ファンクション No. 22

**機能** ページマッピングの変更とジャンプ

**コール** AH = 55H

AL physical page number/segment selector

log\_phys\_mapストラクチャ内のphys\_page\_number\_segメンバの値が物理ページ番号を表しているセグメントなのか、物理ページなのかを示すコード。

AL=0 の場合、物理ページ番号。

AL=1 の場合、物理ページ番号のセグメントアドレス。

DX EMM ハンドル番号

DS : SI map\_and\_jumpストラクチャへのポインタ

要求された物理ページをマッピングし、ターゲットアドレスにジャンプするために必要な情報を含むストラクチャへのポインタ。

```
log_phys_map_struct STRUC
 log_page_number DW ?
 phys_page_number_seg DW ?
log_phys_map_struct ENDS
```

```
map_and_jump_struct STRUC
 target_address DD ?
 log_phys_map_len DB ?
 log_phys_map_ptr DD ?
map_and_jump_struct ENDS
```

**リターン** AH = 00H 正常実行 (指定のエリアに制御が渡った)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 83H 回復不可 (指定の EMM ハンドルがない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8AH 復旧可能 (マッピングする論理ページが EMM ハンドルにアロケートされている論理ページの範囲を超えている)



= 8BH 復旧可能(指定されたマップ可能なセグメントアドレスがマップングできない)

= 8FH 回復不可(サブファンクションパラメータが無効)

### 解説

このファンクションは、メモリへのマップを変更し、指定のアドレスに制御を渡します。これは、8086 ファミリアーキテクチャの FAR JUMP に似ています。このファンクションがコールされる前のメモリマップングコンテキストは、消えてしまいます。

ページをマップングしないでジャンプしても、エラーとは見なされません。もし、ページをマップングしないでジャンプする要求があった場合は、目的のアドレスに制御が渡り、このファンクションは FAR JUMP を実行します。

### ●パラメータの説明

ストラクチャのメンバは以下のとおりです。

#### . target\_address

最初のメンバは、制御が移されるターゲットアドレスを含む FAR ポインタです。アドレスはセグメント：オフセット形式で表現されます。アドレスのオフセット値はダブルワードの下位ワードに格納されます。

#### . log\_phys\_map\_len

2番目のメンバはバイトで、すぐ後に続くストラクチャの配列中のエントリ数を示します。配列の大きさは、要求された論理ページを物理ページにマップングするために必要な長さです。エントリ数は、システム中でマップングできる物理ページの数を超えることはできません。

#### . log\_phys\_map\_ptr

3番目のメンバは、論理ページ番号と、マップングされる物理ページまたはセグメントアドレスを含んでいるストラクチャの配列へのポインタです。ストラクチャの配列中の各エントリは次の2つです。

#### . log\_page\_number

このストラクチャの最初のメンバは、マップングされるべき論理ページの数を含むワードです。

#### . phys\_page\_number\_seg

このストラクチャの2番目のメンバはワードで、最初のメンバの論理ページがマップングされるべき物理ページの番号か、またはセグメントアドレス形式のど

ちらかを含みます。ALに渡される値で、どちらの表現かが決定されます。

例

|                       |                                 |
|-----------------------|---------------------------------|
| log_phys_map          | log_phys_map_struct (?) DUP (?) |
| map_and_jump          | map_and_jump_struct (?)         |
| emm_handle            | DW ?                            |
| phys_page_or_seg_mode | DB ?                            |

```

MOV AX,SEG map_and_jump ; set map and jump segment
MOV DS,AX ;
LEA SI,map_and_jump ; DS:SI map and jump pointer
MOV DX,emm_handle ; set EMM handle
MOV AH,55H ; load function code
MOV AL,phys_page_or_seg_mode ; set phys page or seg mode
INT 67H ; call the memory manager
OR AH,AH ; check EMM status
JNZ emm_err_handler ; jump to error handler on error

```

## INT 67H

## Alter Page Map &amp; Call

## ファンクション No. 23

**機能** ページマッピングの変更とコール

**コール** AH = 56H

AL physical page number/segment selector

log\_phys\_mapストラクチャ内のphys\_page\_number\_segメンバに含まれる値が、物理ページ番号を表すか、物理ページ番号を表しているセグメントアドレスなのかを示す。

AL=0 の場合、物理ページ番号。

AL=1 の場合、物理ページ番号のセグメントアドレス。

DX EMM ハンドル番号

DS:SI map\_and\_callストラクチャへのポインタ

要求された論理ページを指定の物理ページにマッピングし、ターゲットアドレスをコールするために必要な情報を含むストラクチャへのポインタ。

```

log_phys_map_struct STRUC
 log_page_number DW ?
 phys_page_number_seg DW ?
log_phys_map_struct ENDS

map_and_call_struct STRUC
 target_address DD ?
 new_page_map_len DB ?
 new_page_map_ptr DD ?
 old_page_map_len DB ?
 old_page_map_ptr DD ?
 reserved DW 4DUP(?)
map_and_call_struct ENDS

```

**リターン** AH = 00H 正常実行（制御がターゲットアドレスに渡った）  
 = 80H 回復不可（メモリマネージャソフトウェアが動作しない）  
 = 81H 回復不可（拡張メモリハードウェアが動作しない）  
 = 83H 回復不可（指定の EMM ハンドルがない）  
 = 84H 回復不可（メモリマネージャに渡されたファンクションが未定義）



- = 8AH 復旧可能(対応する物理ページにマッピングされる1つ以上の論理ページが、EMM ハンドルにアロケートされている論理ページの範囲を超えた。プログラムは、EMM ハンドルの範囲内で論理ページをマッピングすることにより、復旧できる)
- = 8BH 復旧可能(1つ以上の物理ページが、指定できる物理ページの範囲を超えた。または、システム内に存在する物理ページ数以上のページを指定した。物理ページ番号は、0 から番号づけられます。プログラムは、0 から(システムがサポートしている物理ページ数-1)までの範囲内の物理ページをマッピングすることにより復旧できる)
- = 8FH 回復不可(サブファンクションパラメータが無効)

**解説**

このファンクションは、カレントのマッピングコンテキストをセーブし、指定のマッピングコンテキストを変更し、指定のアドレスに制御を渡します。これは、8086 ファミリアーキテクチャの FAR CALL に似ています。FAR CALL から戻ったときにコードセグメント内の値を元に戻すのと同様に、このファンクションもリターン、指定のマッピングコンテキストを元の状態に戻します。

FAR CALL からのリターンをそのままエミュレートするような拡張メモリサブファンクションはありませんが、FAR CALL の標準的なリターンを実行することでリターンできます。以下で、この機能について説明します。

このファンクションの起動後エラーが検出されなければ、メモリマネージャが、指定されたアドレスに制御を移します。エラーが生じた場合、メモリマネージャはただちに AH レジスタにエラーコードを返しますが、発生しなかった場合、メモリマネージャはリターン後にマッピングコンテキストの状態を復元するために、スタックに情報をプッシュしておきます。

呼び出されたプロシージャが呼び出したプロシージャに値を返す必要がある場合は、単に標準的な FAR RETURN を行います。メモリマネージャはこのリターンをトラップし、退避したマッピングコンテキストを復元し、呼び出したプロシージャにリターンします。メモリマネージャは、他のファンクションと同様に、成功した場合にもステータスを返します。

このサブファンクションを用いる開発者は、このサブファンクションが使用する分のスタックスペースを考慮しなければなりません。

## ●パラメータの説明

ストラクチャのメンバは以下のとおりです。

## . target\_address

最初のメンバは、制御が移されるターゲットアドレスを含む FAR ポインタです。アドレスは、セグメント：オフセットの形式で表現されます。アドレスのオフセット部分は、ポインタの下位ワードに格納されます。アプリケーションプログラムは、この値を供給しなければなりません。

## . new\_page\_map\_len

2番目のメンバはバイトで、new\_page\_map\_ptrが指す新しいマッピングコンテキストのエントリ数を示します。この値は、システム内でマッピング可能なページ数を超えることはできません。

## . new\_page\_map\_ptr

3番目のメンバは、論理ページ番号と、コール後すぐにそれらがマッピングされる物理ページ番号、またはセグメントを含むストラクチャの配列へのFARポインタです。新しいストラクチャ配列の内容は、map\_and\_callストラクチャの後におかれます。

## . old\_page\_map\_len

4番目のメンバはバイトで、old\_page\_map\_ptrが指す古いマッピングコンテキスト内のエントリ数を示します。この値は、システム内でマップ可能なページ数を超えることはできません。

## . old\_page\_map\_ptr

5番目のメンバは、論理ページ番号と、リターン後すぐにそれらがマッピングされる物理ページ番号、またはセグメントを含むストラクチャの配列へのFARポインタです。元のストラクチャの配列内容はmap\_and\_callストラクチャの後におかれます。

## . reserved

6番目のメンバは、メモリマネージャ用に予約されています。

ストラクチャの配列内の各エントリは、以下の2つのメンバです。

## . log\_page\_number

このストラクチャの最初のメンバは、コールやリターン直後に、2番目のメンバで指定される物理ページ番号、またはセグメントアドレス表現に、マッピングする論理ページ番号を表すワードです。

## . phys\_page\_number\_seg

このストラクチャの第2のメンバは、コールやリターン直後に、最初のメンバで指定される論理ページ番号にマッピングされるべき物理ページの番号かセグメントアドレス表現のどちらかを表すワードです。

|   |
|---|
| 例 |
|---|

|                       |                                 |
|-----------------------|---------------------------------|
| new_page_map          | log_phys_map_struct (?) DUP (?) |
| old_page_map          | log_phys_map_struct (?) DUP (?) |
| map_and_call          | map_and_call_struct (?) DUP (?) |
| emm_handle            | DW ?                            |
| phys_page_or_seg_mode | DB ?                            |

|     |                          |                                  |
|-----|--------------------------|----------------------------------|
| MOV | AX,SEG map_and_call      | ; set map and call segment       |
| MOV | DS,AX                    | ;                                |
| LEA | SI,map_and_call          | ; DS:SI map and call pointer     |
| MOV | DX,emm_handle            | ; set EMM handle                 |
| MOV | AH,56H                   | ; load function code             |
| MOV | AL,phys_page_or_seg_mode | ; set phys page or seg mode      |
| INT | 67H                      | ; call the memory manager        |
| OR  | AH,AH                    | ; check EMM status               |
| JNZ | emm_err_handler          | ; jump to error handler on error |



## INT 67H

**Get Page Map Stack Space Size**

ファンクション No. 23  
コード 02H

**機能** ページマップスタックサイズの取得

**コール** AX = 5602H

**リターン** AH = 00H 正常実行 (スタックサイズが返された)  
           = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
           = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
           = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
           = 8FH 回復不可 (サブファンクションパラメータが無効)  
       BX 要求されたスタックスペース

**解説** Alter Page Map & Callファンクションはスタック上に追加情報 (リターンアドレスを含む) をプッシュするので、このサブファンクションは、そのファンクションが要求するスタックスペースのバイト数を返します。

## INT 67H

**Move Memory Region**

**ファンクション No. 24**  
**コード 00H**

**機能**      メモリ領域の移動

**コール**      AX = 5700H

DS : SI move\_source\_dest ストラクチャへのポインタ

移動するための、ソースとデスティネーションの情報を含むストラクチャへのポインタ、

```

move_source_dest_struct STRUC
 region_length DD ?
 source_memory_type DB ?
 source_handle DW ?
 source_initial_offset DW ?
 source_initial_seg_page DW ?
 dest_memory_type DB ?
 dest_handle DW ?
 dest_initial_offset DW ?
 dest_initial_seg_page DW ?
move_source_dest_struct ENDS

```

**リターン**      AH = 00H 正常実行 (メモリ領域の移動を行った)

= 80H 回復不可 (メモリマネージャソフトウェアが動作しない)

= 81H 回復不可 (拡張メモリハードウェアが動作しない)

= 83H 復旧可能 (ソースまたはデスティネーションの EMM ハンドルを見つけられなかった、メモリマネージャは指定のハンドルの情報を得られなかった、EMM ハンドルは無効)

= 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)

= 8AH 回復不可 (指定の論理ページが、EMM ハンドルに割り当てられているページを超えた)

= 8FH 回復不可 (サブファンクションパラメータが無効)

- = 92H 成功(ソースまたはデスティネーションの拡張メモリ領域は、同じ EMM ハンドルを持ち重複している。この場合、移動は行える。移動は完了し、ソース領域は全部デスティネーションにコピーされた。しかし、少なくともソース領域は移動によって重ね書きされてしまった。異なるハンドルを持つソース領域とデスティネーション領域は、それぞれ異なるメモリ領域を指定するので、物理的には重複されることはない)
- = 93H 状況により復旧可能(指定されたソースまたはデスティネーションの拡張メモリの大きさが、ソースまたはデスティネーションハンドルにアロケートされている拡張メモリページの大きさを超えた。このハンドルにアロケートされているページサイズでは、領域のコピーができない。プログラムは、ソースまたはデスティネーションハンドルに追加のページをアロケートし、再度このファンクションを実行することにより、上記の状況から復旧できる。しかし、アプリケーションプログラムが必要としているにも関わらず、すでに拡張メモリをアロケートしていたならば、エラーとなり復旧できない)
- = 94H 回復不可(内部メモリと拡張メモリの領域が重複している。これは無効で、内部メモリは拡張メモリと重複することはできない)
- = 95H 回復不可(論理ページ内のオフセットが、論理ページの大きさを超えた。拡張メモリ領域内の最初のソースまたはデスティネーションのオフセットは、0000H から 3FFFH(16383)、または、[論理ページの大きさ-1] でなくてはならない)
- = 96H 回復不可(領域サイズが 1MB を超えた)
- = 98H 回復不可(メモリのソースとデスティネーションのタイプが定義されていない。またはサポートされていない)
- = A2H 回復不可(移動中に、内部メモリの 1M バイトアドレススペースを超えようとした。ソース/デスティネーションの開始アドレスの組み合わせと、移動されるべき領域の大きさが 1M バイトを超えた。データは移動されなかった)



**解 説**

このファンクションは、以下のメモリのソース/デスティネーションの組み合わせで、メモリ領域のコピーを行います。

- ・内部メモリ → 内部メモリ
- ・内部メモリ → 拡張メモリ
- ・拡張メモリ → 内部メモリ
- ・拡張メモリ → 拡張メモリ

メモリを移動するために、拡張メモリのマッピングコンテキストをセーブ、リストアする必要はありません。カレントのマッピングコンテキストは、このオペレーションを行っても保持されます。

領域の大きさは、指定の EMM ハンドルにアロケートされた拡張メモリページのサイズにより制限されます。しかし、ほとんどの実用的なアプリケーションプログラムでは、領域の大きさは限界サイズよりも小さくなっています。領域サイズが 0 でもエラーではなく、メモリの移動は実行されません。

領域サイズが、16K バイトを超えてもエラーになりません。この場合、論理ページの 1 グループが移動の対象となります。指定の論理ページは、移動が行われる最初の論理ページを表します。もし領域のサイズが 16K バイトを超えた場合、または 16K バイト未満であっても複数の論理ページにまたがっているならば、全体の領域に合わせるために最初の論理ページの後に十分な大きさの論理ページが残っていないてはなりません。

もし、アプリケーションが拡張メモリに内部メモリ領域をセーブする必要があるれば、マッピングコンテキストのセーブまたはリストアを実行しなくても領域を移動することができます。メモリマネージャがこのコンテキストを維持し、最大 1M バイトまでの移動ができます。しかし、実際の移動サイズはこれよりも小さくなるでしょう。

もし、ソース EMM ハンドルとデスティネーション EMM ハンドルが同じならば、ソース領域とデスティネーション領域は、移動の前に重複しているかどうかチェックされます。そして領域が重複していても移動方向が正しく選択され、デスティネーション領域にはソース領域が完全にコピーされ、領域の重複が発生したことを示すステータスが返されます。

## ●パラメータの説明

ストラクチャのメンバは以下のとおりです。

## . region\_length

最初のメンバはダブルワードで、移動されるメモリ領域の大きさ（バイト）を指定します。

## . source\_memory\_type

2番目のメンバはバイトで、ソース領域のメモリのタイプを指定します。これが0ならば、ソース領域は内部メモリ内（ページフレームセグメントを除く）に存在し、1ならば拡張メモリ内に存在することを示します。

## . source\_handle

もし、ソース領域が拡張メモリ内にあるならば、3番目のメンバはソースメモリ領域と関連するEMMハンドル番号を指定するワードです。また、ソース領域が内部メモリ内にあるならば、この変数は意味がなく、将来の互換性のために0にセットしなければなりません。

## . source\_initial\_offset

4番目のメンバはワードで、移動を開始するソース領域内のオフセットを指定します。もし、ソース領域が拡張メモリ内にあるならば、source\_initial\_offsetは16Kバイトの論理ページの最初のアドレスを基準にするので、オフセットは0000Hから3FFFHまでの値をとります。また、ソース領域が内部メモリ内にあるならば、source\_initial\_offsetは移動を開始するソースセグメントの最初のオフセットを指定します。このオフセットは64Kバイトの内部メモリの最初のアドレスを基準にしているため、この値は0000HからFFFFHまでの値をとります。

## . source\_initial\_seg\_page

5番目のメンバはワードで、移動を開始するソース領域のセグメントまたは論理ページ番号を指定します。もし、ソース領域が拡張メモリ内にあれば、移動を開始するソース領域の論理ページを指定します。また、ソース領域が内部メモリ内にあるならば、source\_initial\_seg\_pageは移動を開始する内部メモリの最初のセグメントアドレスを指定します。

## . dest\_memory\_type

6番目のメンバはバイトで、デスティネーション領域が存在するメモリのタイプを指定します。0ならば内部メモリで、1ならば拡張メモリに存在することを示します。



**.dest\_handle**

もし、デスティネーション領域が拡張メモリ内にあるならば、7番目のメンバはデスティネーションメモリ領域に関連する EMM ハンドル番号を指定するワードです。また、デスティネーション領域が内部メモリ内にあるならば、この変数は意味がなく、将来の互換性のために 0 にセットしなければなりません。

**.dest\_initial\_offset**

8番目のメンバはワードで、移動を開始するデスティネーション領域内のオフセットを指定します。もし、デスティネーション領域が拡張メモリ内にあるならば、dest\_initial\_offset は 16K バイトの論理ページの最初のアドレスを基準にするので、オフセットは 0000H から 3FFFH までの値をとります。また、デスティネーション領域が内部メモリ内にあるならば、dest\_initial\_offset は移動を開始するデスティネーションセグメントの最初のオフセットを指定します。このオフセットは 64K バイトの内部メモリの最初のアドレスを基準にしているため、この値は 0000H から FFFFH までの値をとります。

**.dest\_inital\_seg\_page**

9番目のメンバーはワードで、移動を開始するデスティネーション領域のセグメントまたは論理ページ番号を指定します。もし、デスティネーション領域が拡張メモリ内にあれば、移動を開始するデスティネーション領域の論理ページを指定します。また、デスティネーション領域が内部メモリ内にあるならば、dest\_initial\_seg\_page は移動を開始する内部メモリの最初のセグメントアドレスを指定します。

|   |
|---|
| 例 |
|---|

move\_source\_dest

move\_source\_dest\_struct (?)

|     |                         |                                  |
|-----|-------------------------|----------------------------------|
| MOV | AX,SEG move_source_dest | ; set move_source_dest segment   |
| MOV | DS,AX                   | ;                                |
| LEA | SI,move_source_dest     | ; DS:SI move_source_dest pointer |
| MOV | AX,5700H                | ; load function code             |
| INT | 67H                     | ; call the memory manager        |
| OR  | AH,AH                   | ; check EMM status               |
| JNZ | emm_err_handler         | ; jump to error handler on error |



## INT 67H

## Exchange Memory Region

ファンクション No. 24  
コード 01H

**機能** メモリ領域の交換

**コール** AX = 5701H

DS : SI exchange\_source\_dest ストラクチャへのポインタ

交換するためのソースとデスティネーションの情報を含む、ストラクチャへのポインタ。

```

xchg_source_dest_struct STRUC
 region_length DD ?
 source_memory_type DB ?
 source_handle DW ?
 source_initial_offset DW ?
 source_initial_seg_page DW ?
 dest_memory_type DB ?
 dest_handle DW ?
 dest_initial_offset DW ?
 dest_initial_seg_page DW ?
xchg_source_dest_struct ENDS

```

**リターン**

- AH = 00H 正常実行 (メモリ領域の変換が行われた)
- = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)
- = 81H 回復不可 (拡張メモリハードウェアが動作しない)
- = 83H 復旧可能 (ソースまたはデスティネーションの EMM ハンドルを見つけられなかった、メモリマネージャは指定のハンドルの情報を得られなかった、EMM ハンドルは無効)
- = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)
- = 8AH 回復不可 (EMM ハンドルに割り当てられているページ数より大きいページが指定された)
- = 8FH 回復不可 (サブファンクションパラメータが無効)

- = 93H 状況により復旧可能（指定されたソースまたはデスティネーションの拡張メモリの大きさが、ソースまたはデスティネーションハンドルにアロケートされている拡張メモリページの大きさを超えた。このハンドルにアロケートされているページサイズでは、領域のコピーができない。プログラムはソースまたはデスティネーションハンドルに追加のページをアロケートし、再度このファンクションを実行することにより、この状況から復旧できる。しかし、アプリケーションプログラムが必要としているにも関わらず、すでに拡張メモリをアロケートしていたならば、エラーとなり復旧できない）
- = 94H 回復不可（内部メモリと拡張メモリの領域が重複している。これは無効で、内部メモリは拡張メモリと重複することはできない）
- = 95H 回復不可（論理ページ内のオフセットが、論理ページの大きさを超えた。拡張メモリ領域内の最初のソースまたはデスティネーションのオフセットは、0000H から 3FFFH(16383)、または、〔論理ページの大きさ-1〕でなくてはならない）
- = 96H 回復不可（領域サイズが 1M バイトを超えた）
- = 97H 回復不可（ソースとデスティネーションの拡張メモリ領域が同じハンドルを持ち、領域が重複している。これは無効で、ソースとデスティネーションの拡張メモリ領域が交換される場合は、同じハンドルを持って、重複することはできない。異なるハンドルを持つソースとデスティネーションの拡張メモリ領域は、普通異なる拡張メモリ領域を指定するので物理的には決して重複することはない）
- = 98H 回復不可（メモリのソースとデスティネーションのタイプが定義されていない、またはサポートされていない）
- = A2H 回復不可（移動中に内部メモリの 1M バイトアドレススペースを超えようとした。ソース/デスティネーション開始アドレスの組み合わせと交換されるべき領域の大きさが 1M バイトを超えた。データは交換されなかった）

**解 説**

このファンクションは、以下のメモリのソース/デスティネーションの組み合わせにおいて、メモリ領域の交換（ストリング転送）を行います。

- ・内部メモリ → 内部メモリ
- ・内部メモリ → 拡張メモリ
- ・拡張メモリ → 内部メモリ
- ・拡張メモリ → 拡張メモリ



拡張メモリ領域は、640K バイト (9FFFFH) 以上のメモリエリアのみを指します。(PC-9800 シリーズでは 1M バイト (100000H) 以上を指します) もしシステムがマップ可能な内部メモリを備えているならば、このファンクションはマッピング可能な内部メモリを普通の内部メモリとして扱います。ソース領域の内容とデスティネーション領域の内容は交換されます。

交換のオペレーションを行うために拡張メモリのマッピングコンテキストをセーブ、リストアする必要はありません。カレントのマッピングコンテキストは、このオペレーションの間保持されます。領域の大きさは、指定の EMM ハンドルにアロケートされている拡張メモリページのサイズにより制限されます。また、領域の大きさが0でもエラーにならず、交換も実行されません。16K バイトを超える領域サイズもエラーにはなりません。この場合、このファンクションは論理ページの1グループが、交換の対象になります。

指定された論理ページは、交換が実行される最初の論理ページを表しています。

もし、領域の大きさが 16K バイトを超えるとき、または領域のサイズが 16K バイト未満であっても論理ページにまたがっているなら、全体の領域に合わせるために最初の論理ページの後に十分な大きさの論理ページが残っていないてはなりません。

アプリケーションプログラムが、もし、拡張メモリと内部メモリを交換する必要があるならば、カレントのマッピングコンテキストをセーブまたはリストアせずに対象の領域を交換することができます。交換のオペレーションが実行される前に、領域の重複がないかどうかチェックを行う必要があります。交換におけるソースとデスティネーションとの領域の重複は無効であり、交換は実行されません。

#### ●パラメータの説明

ストラクチャのメンバは以下のとおりです。

##### . region\_length

最初のメンバはダブルワードで、交換されるメモリ領域の大きさ (バイト) を指定します。

##### . source\_memory\_type

2 番目のメンバはバイトで、ソース領域のメモリのタイプを指定します。これが 0 ならば、ソース領域は内部メモリ内 (ページフレームセグメントを含む) に存在し、1 ならば拡張メモリ内に存在することを示します。



**. source\_handle**

もし、ソース領域が拡張メモリ内にあるならば、3番目のメンバはソースメモリ領域と関連するハンドル番号を指定するワードです。また、ソース領域が内部メモリ内にあるならば、この変数は意味がなく、将来の互換性のために0をセットしなければなりません。

**. source\_initial\_offset**

4番目のメンバはワードで、交換を開始するソース領域内のオフセットを指定します。もし、ソース領域が拡張メモリ内にあるならば、source\_initial\_offsetは16Kバイトの論理ページの最初のアドレスを基準にするので、オフセットは0000Hから3FFFHまでの値をとります。また、ソース領域が内部メモリ内にあるならば、source\_initial\_offsetは交換を開始するソースセグメントの最初のオフセットを定義します。このオフセットは64KBの内部メモリの最初のアドレスを基準にしているため、この値は0000HからFFFFHまでの値をとります。

**. source\_initial\_seg\_page**

5番目のメンバはワードで、交換を開始するソース領域のセグメントまたは論理ページ番号を指定します。もし、ソース領域が拡張メモリ内にあるならば、交換を開始するソース領域内の論理ページを定義します。また、ソース領域が内部メモリ内にあるならば、source\_initial\_seg\_pageは交換を開始する内部メモリ内の最初のセグメントアドレスを指定します。

**. dest\_memory\_type**

6番目のメンバはバイトで、デスティネーション領域のメモリのタイプを指定します。0ならば内部メモリで、1ならば拡張メモリに存在することを示します。

**. dest\_handle**

もし、デスティネーション領域が拡張メモリ内にあるならば、7番目のメンバはデスティネーションメモリ領域と関連するハンドル番号を指定するワードです。また、デスティネーション領域が内部メモリ内にあるならば、この変数は意味がなく、将来の互換性のために0にセットしなければなりません。

**. dest\_initial\_offset**

8番目のメンバはワードで、交換を開始するデスティネーション領域内のオフセットを指定します。もし、デスティネーション領域が拡張メモリ内にあるならば、destination\_initial\_offsetは16Kバイトの論理ページの最初のアドレスを基準にするので、オフセットは0000Hから3FFFHまでの値をとります。また、デスティネーション領域が内部メモリ内にあるならば、destination\_initial\_offset

は交換を開始するデスティネーションセグメントの最初のオフセットを定義します。このオフセットは 64K バイトの内部メモリの最初のアドレスを基準にしているので、この値は 0000H から FFFFH までの値をとります。

#### .dest\_initital\_seg\_page

9 番目のメンバーはワードで、交換を開始するデスティネーション領域のセグメントまたは論理ページ番号を指定します。もし、デスティネーション領域が拡張メモリ内にあれば、交換を開始するデスティネーション領域の論理ページを定義します。また、デスティネーション領域が内部メモリ内にあるならば、dest\_initial\_seg\_page は交換を開始する内部メモリの最初のセグメントアドレスを指定します。

例

| xchg_source_dest            | xchg_source_dest_struct (?)      |
|-----------------------------|----------------------------------|
| MOV AX,SEG xchg_source_dest | ; set xchg_source_dest segment   |
| MOV DS,AX                   | ;                                |
| LEA SI,xchg_source_dest     | ; DS:SI xchg_source_dest pointer |
| MOV AX,5701H                | ; load function code             |
| INT 67H                     | ; call the memory manager        |
| OR AH,AH                    | ; check EMM status               |
| JNZ emm_err_handler         | ; jump to error handler on error |



## INT 67H

## Get Mappable Physical Address Array

ファンクション No. 25  
コード 00H

**機能** マッピング可能な物理アドレス配列の取得

**コール** AX = 5800H

ES : DI mappable\_phys\_page

メモリマネージャが物理アドレス配列をコピーするアプリケーションプログラムが供給しているメモリエリアへのポインタ。

```
mappable_phys_page_struct STRUC
 phys_page_segment DW ?
 phys_page_number DW ?
mappable_phys_page_struct ENDS
```

**リターン** AH = 00H 正常実行 (物理ページのセグメントアドレス配列が返された)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 83H 回復不可 (指定の EMM ハンドルがない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (サブファンクションパラメータが無効)

CX mappable\_phys\_page 内のエントリ数

物理ページアドレス配列が要求しているバイト数を決めるためには、mappable\_phys\_page\_struct のサイズにこの数値を掛ける。

**解説** このファンクションは、システム内のマッピング可能な各物理ページに対する物理ページ番号と、セグメントアドレスを含む配列を返します。この配列は、システム内の各マップ可能な各物理ページ番号と実際のセグメントアドレスとの間の、クロスリファレンスを与えるものです。また、この配列ではセグメントが昇順に並んでいます。このことは、セグメントアドレスに対応する物理ページも昇順に並んでいることを意味しているわけではありません。

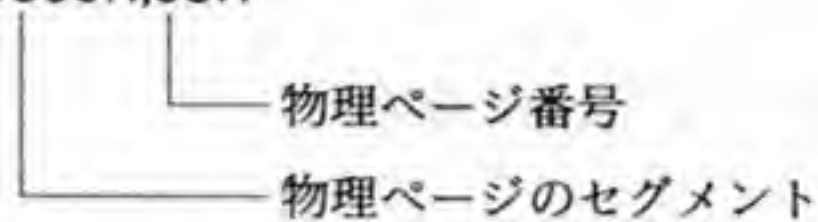


(例) B000H,00H

B400H,01H

B800H,02H

BC00H,03H



## ●パラメータの説明

配列内の各エントリは、次のような2つのメンバを持つストラクチャです。

## . phys\_page\_segment

1番目のメンバはワードで、後に続く物理ページ番号に対応するマップ可能な物理ページのセグメントアドレスです。配列のエントリは、昇順に並んだセグメントアドレスです。

## . phys\_page\_number

2番目のメンバはワードで、前のセグメントアドレスに対応する物理ページ番号です。物理ページ番号は、昇順に並んでいるとは限りません。

例

mappable\_phys\_page

mappable\_phys\_page\_struct (?)

mappable\_page\_entry\_count

DW ?

```

MOV AX,SEG mappable_phys_page ; set mappable phys page segment
MOV ES,AX ;
LES DI,mappable_phys_page ; ES:DI mappable phys page pointer
MOV AX,5800H ; load function code
INT 67H ; call the memory manager
OR AH,AH ; check EMM status
JNZ emm_err_handler ; jump to error handler on error
MOV mappable_page_entry_count,CX
 ; save mappable page entry count

```

## INT 67H

|                                             |                           |
|---------------------------------------------|---------------------------|
| Get Mappable Physical Address Array Entries | ファンクション No. 25<br>コード 01H |
|---------------------------------------------|---------------------------|

**機能** マッピング可能な物理アドレス配列エントリの取得

**コール** AX = 5801H

**リターン** AH = 00H 正常実行 (物理アドレス配列のエントリを返した)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (サブファンクションパラメータが無効)

CX mappable\_phys\_page のエントリ数

このエントリ数は、システム内のマッピング可能な物理ページ数をも示す。物理ページアドレス配列が必要としているバイト数を決めるためには、mappable\_phys\_page\_struct のサイズにこの数値を掛けます。

**解説** このファンクションは、最初のファンクション 25 (Get Mappable Physical Address Array) が返す配列に必要とされるエントリ数を返します。

|          |                              |                                  |
|----------|------------------------------|----------------------------------|
| <b>例</b> | mappable_page_entry_count    | DW ?                             |
| MOV      | AX,5801H                     | ; load function code             |
| INT      | 67H                          | ; call the memory manager        |
| OR       | AH,AH                        | ; check EMM status               |
| JNZ      | emm_err_handler              | ; jump to error handler on error |
| MOV      | mappable_page_entry_count,CX |                                  |
|          |                              | ; save mappable page entry count |

## INT 67H

## Get Hardware Configuration Array

ファンクション No. 26  
コード 00H

**機能** ハードウェア構成配列の取得

**コール** AX = 5900H

ES : DI hardware\_info

メモリマネージャが拡張メモリハードウェア情報をコピーする、OS が供給しているメモリへのポインタ。

```

hardware_info_struct STRUC
 raw_page_size DW ?
 alternate_register_sets DW ?
 context_save_ares_size DW ?
 DMA_register_sets DW ?
 DMA_channel_operation DW ?
hardware_info_struct ENDS

```

**リターン** AH = 00H 正常実行 (ハードウェア構成配列が返された)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 83H 回復不可 (指定の EMM ハンドルがない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (サブファンクションパラメータが無効)  
 = A4H 回復不可 (このファンクションにアクセスすることを OS が禁じている、この場合、このファンクションは使用できない)

hardware\_info 拡張メモリハードウェアの情報

**解説** このファンクションは、OS/E 環境により使用される拡張メモリのハードウェア構成情報を含む配列を返します。

**注意:** このファンクションは、OS のみが使用できます。このファンクションは OS により、いつでも使用禁止にすることが可能です。その方法については、ファンクション 30 の解説を参照してください。



### ●パラメータの説明

ストラクチャは、以下の5つのメンバを持っています。

#### . raw\_page\_size

1番目のメンバはワードで、マッピング可能なロー物理ページのサイズをパラグラフ単位(16 バイト)で示します。EMSの標準ページは16K バイトです。しかし他の拡張メモリボードの動作ではこの標準サイズに従っているわけではなく、複数のより小さなページをマッピングすることにより16K バイトのページをエミュレートできます。このメンバは、ハードウェアの動作レベルから見たマップ可能なページのサイズを指定するものです。

注) PC-9800 シリーズでのローページのサイズは16 KB です。

#### . alternate\_register\_sets

2番目のメンバはワードで、変更するマッピングレジスタのセットの数が入ります。追加のマッピングレジスタセットを、このマニュアルでは代替マッピングレジスタセットと呼びます。すべての拡張メモリボードは、論理ページから物理ページへのマッピングを実行するために、少なくとも1つのハードウェアレジスタのセットを持っています。また、拡張メモリボードの中には、1つ以上のマッピングレジスタを持っているものもあります。このメンバは、システム内にいくつの代替マッピングレジスタセットが存在するかが入ります(すべての拡張メモリが持っている1セットは除く)。もし拡張メモリボードがマッピングレジスタのセットを1つしか持っていない場合(つまり代替マッピングレジスタのセットがない場合)、このメンバの値は0です。

#### . context\_save\_area\_size

3番目のメンバはワードで、マッピングコンテキストをセーブするために必要な配列の大きさが入ります。このメンバに返される値は、ファンクション15(Get Size of Page Map Save Array)で返される値とまったく同じです。

#### . DMA\_register\_sets

4番目のメンバはワードで、DMA チャンネルに割り当てられるレジスタセットの数が入ります。このDMA レジスタセットは、代替レジスタセットの使用法と似ていますが、DMA のマッピング用であり、タスクのマッピング用ではありません。もし、拡張メモリハードウェアが、DMA レジスタセットをサポートしていなければ、DMA が実行されるときは注意しなければなりません。マルチタスク

OSでは、あるタスクがDMAが完了するのを待っている場合、別のタスクにスイッチを切り換えるのに便利です。しかし、次のタスクがリマッピングするための必要なメモリをDMAが操作していた場合は、リマッピングの結果は保証されません。また、拡張メモリハードウェアがDMAの動作状態を感知できるならば、OS/EはDMAの間の、タスク切り換えとリマッピングを許可すべきです。DMAの特別のサポートがなければ、DMA実行中にはリマッピングは行うべきではありません。

#### . DMA\_channel\_operation

5番目のメンバはワードで、DMAレジスタセット用の特別な場合が入ります。この値が0ならば、DMAレジスタセットはファンクション28で記述してあるものと同じように機能します。また、1ならば、拡張メモリハードウェアはDMAレジスタセットを1つだけしか所有していません。さらに、もしどれかのチャンネルがこのレジスタセットを通じてマップされた場合、全チャンネルはこのレジスタを通じてマップされます。EMS標準ボードでは、この値は0です。

| 例 | hardware_info            | hardware_info_struct (?)         |
|---|--------------------------|----------------------------------|
|   | MOV AX,SEG hardware_info | ; set hardware info segment      |
|   | MOV ES,AX                | ;                                |
|   | LEA DI,hardware_info     | ; ES:DI hardware info offset     |
|   | MOV AX,5900H             | ; load function code             |
|   | INT 67H                  | ; call the memory manager        |
|   | OR AH,AH                 | ; check EMM status               |
|   | JNZ emm_err_handler      | ; jump to error handler on error |



## INT 67H

**Get Unallocated Raw Page Count**

**ファンクション No. 26**  
**コード 01H**

**機能** 未アロケートのローページカウンタの取得

**コール** AX = 5901H

**リターン** AH = 00H 正常実行 (未アロケートローページ数とローページの総数を返した)

= 80H 回復不可 (メモリマネージャソフトウェアが動作しない)

= 81H 回復不可 (拡張メモリハードウェアが動作しない)

= 83H 回復不可 (指定の EMM ハンドルがない)

= 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)

= 8FH 回復不可 (サブファンクションパラメータが無効)

**BX** 未アロケートのローページ数

現在使用可能なローページの数。

**DX** ローページの総数

拡張メモリ内のローページの総数。

**解説** このファンクションは、OS に対する拡張メモリ内の (標準サイズではない) マップ可能なページ総数と、アロケートされていない (標準サイズではない) マップ可能なページ数を返します。

ある種類の拡張メモリボードは、16K バイトの約数となるようなページサイズを持つものがあり、16K バイトの約数となるような拡張メモリページはローページと呼ばれます。OS は 16K バイトの約数となるようなマップ可能な物理ページを処理することもあります。

もし、拡張メモリボードが、ちょうど 16K バイトの倍数サイズのページを供給しているとすれば、このファンクションが返す数はファンクション 3 (Get Unallocated Page Count) が返す値と同じになります。

注) PC-9800 シリーズでは、標準ページとローページは同じサイズ (16 K バイト) です。



|   |                          |                                  |
|---|--------------------------|----------------------------------|
| 例 | unalloc_raw_pages        | DW ?                             |
|   | total_raw_pages          | DW ?                             |
|   | MOV AX,5901H             | ; load function code             |
|   | INT 67H                  | ; call the memory manager        |
|   | OR AH,AH                 | ; check EMM status               |
|   | JNZ emm_err_handler      | ; jump to error handler on error |
|   | MOV unalloc_raw_pages,BX | ; save unalloc raw pages         |
|   | MOV total_raw_pages,DX   | ; save total raw pages           |

## INT 67H

## Allocate Standard Pages

ファンクション No.27  
コード 00H

- 機能** 標準サイズのページのアロケートと固有の EMM ハンドルの割り当て
- コール** AX = 5A00H  
BX num\_of\_standard\_pages\_to\_alloc  
オペレーティングシステムがアロケートしようとする標準ページ数
- リターン** AH = 00H 正常実行 (メモリマネージャが割り当てられた EMM ハンドルにページをアロケートした)  
= 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
= 81H 回復不可 (拡張メモリハードウェアが動作しない)  
= 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
= 85H 復旧可能 (すべての EMM ハンドルが使用されている)  
= 87H 復旧可能 (システム内にオペレーティングシステムの要求を満たす数の拡張メモリページがない)  
= 88H 復旧可能 (システム内にオペレーティングシステムの要求を満たす数の未アロケートページがない)  
= 8FH 回復不可 (サブファンクションパラメータが無効)

## DX EMM ハンドル

固有の EMM ハンドル。オペレーティングシステムにおいては、この EMM ハンドルをパラメータとして必要とするすべてのファンクションで、このハンドルを使用しなければならない。255個までの EMM ハンドルを使用できる (ファンクション 27 と ファンクション 4 は、同じ 255 個のハンドルを共有しなければならない)。この EMM ハンドルを使用するすべてのファンクションについて、これにアロケートされている物理および論理ページの長さは、標準サイズ (16K バイト) である。

- 解説** このファンクションは、オペレーティングシステムが要求する数だけ、標準サイズ (16K バイト) のページをアロケートし、各ページに固有の EMM ハンドルを割り当てます。この EMM ハンドルは、オペレーティングシステムが EMM ハンドルを解除するまで、これらのページを処理します。このファンクションは、ファンクション 4 (Allocate Pages ファンクション) とは異なり、EMM ハンドルにゼロページをアロケートすることができます。

**注意：**以下の注意は、拡張メモリマネージャインプリメンタとオペレーティングシステムの開発者にのみ関係があります。

アプリケーションは、次に挙げるようなメモリマネージャの特性を使用すべきではありません。アプリケーションがこれらの規則を守らない場合には、マイクロソフト社のオペレーティングシステムに互換性がなくなります。

拡張メモリマネージャは、この仕様に互換性をもたせるために、オペレーティングシステムでのみ使用可能な特別のハンドルを供給しています。このハンドルは、0000H の値を持ち、拡張メモリマネージャドライバがインストールされたときに、そこにアロケートされたページセットを与えられます。メモリマネージャが自動的にEMMハンドル 0000H にアロケートするページは、標準メモリを埋めなおします。一般的には、この埋めなおしは 40000H(256K)と 9FFFFH(640K)の間で起こります。しかし、ハードウェアとメモリマネージャにその能力があれば、この制限より上回ったり下回ったりすることができます。

オペレーティングシステムは、このハンドルを取得するために、ファンクション 27 を呼び出す必要はありません。拡張メモリデバイスドライバがインストールされると、このハンドルはすでに存在していると想定され、ただちに使用可能になるからです。オペレーティングシステムがこのハンドルを使用するときには、0000H の特別な EMM ハンドル値を使用します。オペレーティングシステムは、この特別な EMM ハンドル値を使用して、任意の EMM ファンクションを呼び出すことができます。この EMM ハンドルにページをアロケートするためには、オペレーティングシステムはファンクション 18 (Reallocate Pages ファンクション) を呼び出します。

このハンドルには、2つの特別な場合があります。

#### 1. ファンクション 27 (Allocate Standard Pages ファンクション)

このファンクションは、EMM ハンドル値として 0 (ゼロ) を返すことはありません。アプリケーションにおいて、ページをアロケートし、そのページを持つ EMM ハンドルを取得するためには、ファンクション 27 を呼び出さなければなりません。ファンクション 27 は、ゼロの EMM ハンドル値を返すことはないので、アプリケーションでこの特別な EMM ハンドルにアクセスすることはできません。

#### 2. ファンクション 6 (Deallocate Pages ファンクション)

オペレーティングシステムが、特別な EMM ハンドルにアロケートされたペー



ジの解除のためにこのファンクションを使用すると、EMM ハンドルが所有するページは使用可能としてメモリマネージャに返されます。しかし、この EMM ハンドルは再び割り当てすることはできません。メモリマネージャは、Deallocate Pages ファンクションのこの EMM ハンドルへの要求を、Reallocate Pages ファンクションの要求と同じものとして取り扱います。すなわち、この EMM ハンドルへのリアロケートのページ数はゼロになります。

|   |
|---|
| 例 |
|---|

|                                |      |
|--------------------------------|------|
| num_of_standard_pages_to_alloc | DW ? |
|--------------------------------|------|

|             |      |
|-------------|------|
| emmm_handle | DW ? |
|-------------|------|

|     |                                   |  |
|-----|-----------------------------------|--|
| MOV | BX,num_of_standard_pages_to_alloc |  |
|-----|-----------------------------------|--|

|     |          |                     |
|-----|----------|---------------------|
| MOV | AX,5A00H | ;load function code |
|-----|----------|---------------------|

|     |     |                          |
|-----|-----|--------------------------|
| INT | 67H | ;call the memory manager |
|-----|-----|--------------------------|

|    |       |                   |
|----|-------|-------------------|
| OR | AH,AH | ;check EMM status |
|----|-------|-------------------|

|     |                  |                                 |
|-----|------------------|---------------------------------|
| JNS | emmm_err_handler | ;jump to error handler on error |
|-----|------------------|---------------------------------|

|     |                |              |
|-----|----------------|--------------|
| MOV | emmm_handle,DX | ;save handle |
|-----|----------------|--------------|

## INT 67H

## Allocate Raw Pages

ファンクション No.27  
コード 01H

- 機能** ローページのアロケートと固有の EMM ハンドルの割り当て
- コール** AX = 5A01H  
BX num\_of\_raw\_pages\_to\_alloc  
オペレーティングシステムがアロケートしようとするローページ数
- リターン** AH = 00H 正常実行 (メモリマネージャが割り当てられた EMM ローハンドルにローページをアロケートした)  
= 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
= 81H 回復不可 (拡張メモリハードウェアが動作しない)  
= 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
= 85H 復旧可能 (すべての EMM ハンドルが使用されている)  
= 87H 復旧可能 (システム内にオペレーティングシステムの要求を満たす数の拡張メモリローページがない)  
= 88H 復旧可能 (システム内にオペレーティングシステムの要求を満たす数の未アロケートローページがない)  
= 8FH 回復不可 (サブファンクションパラメータが無効)

## DX EMM ローハンドル

固有の EMM ローハンドル。オペレーティングシステムにおいては、この EMM ローハンドルをパラメータとして必要とするすべてのファンクションで、このハンドルを使用しなければならない。255 個までの EMM ハンドルを使用できる (ファンクション 27 とファンクション 4 は、同じ 255 個の EMM ハンドルを共有しなければならない)。この EMM ローハンドルを使用するすべてのファンクションについて、これにアロケートされている物理および論理ページの長さは、標準サイズ (16K バイト) ではない。

- 解説** このファンクションは、オペレーティングシステムが要求する数だけ、標準サイズ (16K バイト) ではないページ (ローページ) をアロケートし、各ページに固有の EMM ハンドルを割り当てます。この EMM ハンドルは、オペレーティングシステムが EMM ハンドルを解除するまで、これらのページを処理します。このファンクションは、ファンクション 4 (Allocate Pages ファンクション) とは異なり、EMM ハンドルにゼロページをアロケートすることができます。  
(注: PC-9800 シリーズのローページは 16K バイトサイズです)



ハードウェアによっては、16K バイトの約数になるページサイズをもつ拡張メモリボードがあります。16K バイトの約数となる物理ページサイズは、ロー (raw) ページと呼ばれます。オペレーティングシステムが、この 16k バイトの約数となるページサイズを処理する場合もあるでしょう。そのような場合には、メモリマネージャは、ローページがアロケートされたハンドルを使用して、すべてのファンクションを取り扱わなければなりません。これは、Allocate Raw Pages ファンクションを使用して処理を行いますが、標準 (16K バイト) のページがアロケートされている EMM ハンドルを用いる場合とは、異なる使い方をします。

ファンクション 4 (Allocate Pages ファンクション) や、ファンクション 27 (Allocate Standard Pages サブファンクション) を使用して割り当てられた EMM ハンドルは、16K バイトのページを持っていなければなりません。これが標準的な拡張メモリページの大きさです。もし、拡張メモリボードハードウェアが 16K バイトのページを用意できなければ、メモリマネージャは、複数の標準サイズではないページを組み合わせて 1 つが 16K バイトのページを形成することで、16K バイトのページをエミュレートしなければなりません。

ファンクション 27 (Allocate Raw Pages ファンクション) を使用して割り当てられた EMM ハンドルは、ローハンドルと呼ばれます。ローハンドルにアロケートされる論理ページは、すべて標準的でない大きさ (16K ではないサイズ) になります。しかし、いったんオペレーティングシステムが、ハンドルに複数のローページをアロケートしてしまうと、そのローハンドルに標準サイズではないページがアロケートされていることを識別することは、メモリマネージャの役割となります。メモリマネージャは、これらのハンドルを識別し、標準サイズではないページをもつハンドルを使用するすべてのファンクションを、別々に取り扱わなければなりません。論理ページサイズはファンクション 27 が割り当てているいずれかの EMM ローハンドルについても、標準サイズではないページとなります。

**注意：**以下の注意は、拡張メモリマネージャインプリメンターとオペレーティングシステムの開発者にのみ関係があります。

アプリケーションプログラムは、次に挙げるようなメモリマネージャの特性を使用するべきではありません。アプリケーションプログラムがこれらの規則を守らない場合には、マイクロソフト社のオペレーティングシステムに互換性がなくなります。



拡張メモリマネージャは、この仕様に互換性をもたせるために、オペレーティングシステムでのみ使用可能な特別の EMM ハンドルを供給しています。この EMM ハンドルは、0000H の値を持ち、拡張メモリマネージャドライバがインストールされたときに、そこにアロケートされたページセットを与えられます。メモリマネージャが自動的にハンドル 0000H にアロケートするページは、標準メモリを埋めなおします。一般的には、この埋めなおしは 40000H (256K) と 9FFFFH (640K) の間で起こります。しかし、ハードウェアとメモリマネージャにその能力があれば、この制限より上回ったり下回ったりすることができます。

オペレーティングシステムは、この EMM ハンドルを取得するために、ファンクション 27 を呼び出す必要はありません。拡張メモリデバイスドライバがインストールされると、この EMM ハンドルはすでに存在していると想定され、ただちに使用可能になるからです。オペレーティングシステムがこのハンドルを使用するときには、0000H の特別な EMM ハンドル値を使用します。オペレーティングシステムは、この特別なハンドル値を使用して、任意の EMM ファンクションを呼び出すことができます。この EMM ハンドルにページをアロケートするためには、オペレーティングシステムはファンクション 18 (Reallocate Pages ファンクション) を呼び出します。

このハンドルには、2つの特別な場合があります。

#### 1. ファンクション 27 (Allocate Raw Pages ファンクション)

このファンクションは、ハンドル値として 0 (ゼロ) を返すことはありません。アプリケーションにおいて、ページをアロケートし、そのページを持つ EMM ハンドルを取得するためには、ファンクション 27 を呼び出さなければなりません。ファンクション 27 は、ゼロの EMM ハンドル値を返すことはないので、アプリケーションでこの特別な EMM ハンドルにアクセスすることはできません。

#### 2. ファンクション 6 (Deallocate Pages ファンクション)

オペレーティングシステムが、特別な EMM ハンドルにアロケートされたページの解除のためにこのファンクションを使用すると、EMM ハンドルが所有するページは使用可能としてメモリマネージャに返されます。しかし、この EMM ハンドルは再び割り当てすることはできません。メモリマネージャは、Deallocate Pages ファンクションのこの EMM ハンドルへの要求を、Reallocate Pages ファンクションの要求と同じものとして取り扱います。すなわち、この EMM ハンドルへのリアロケートのページ数はゼロになります。

|   |
|---|
| 例 |
|---|

|                                  |                                 |
|----------------------------------|---------------------------------|
| num_of_raw_pages_to_alloc        | DW ?                            |
| emm_raw_handle                   | DW ?                            |
| MOV BX,num_of_raw_pages_to_alloc |                                 |
| MOV AX,5A01H                     | ;load function code             |
| INT 67H                          | ;call the memory manager        |
| OR AH,AH                         | ;check EMM status               |
| JNS emm_err_handler              | ;jump to error handler on error |
| MOV emm_raw_handle,DX            | ;save raw handle                |

## INT 67H

## Alternate Map Register Set

## ファンクション No. 28

**機 能** マップレジスタの変更

**注 意** このファンクションは OS のみを使用します。OS はいつでもこのファンクションを使用禁止にすることができます。OS がこのファンクションを使用可または使用不可にする方法についてはファンクション 30 を参照してください。

●設計構想

記述されているサブファンクションをサポートしているハードウェアは、どんな拡張メモリボード上にもあるとは限りません。あるサブファンクションにはソフトウェアエミュレーションが用意されていて、また他のサブファンクションには、使用に際して一定のプロトコルが実行されなくてはならなかったりします。システム DMA 機能を利用するサブファンクションは、特にこのことが重要になります。

●システム DMA 機能と DMA による拡張メモリサポート

マルチタスク OS では、あるタスクが DMA が完了するのを待っている場合、別なタスクに切り換えできれば便利です。この仕様は DMA が発生している間マップアウトされているメモリ領域に DMA できるように拡張メモリボードに設計されている機能について記述します。この機能を備えていない拡張メモリボードは、マップ可能なメモリ領域で DMA を実行中、マッピングコンテキストを変更することはできません。つまり、すべての DMA 動作は、ページが再マップされる前に完了しておかなくてはなりません。

●拡張メモリによる DMA レジスタセットのサポート

DMA レジスタセットを持つ拡張メモリボードは、マッピングコンテキストが切り換えられている間、マップ可能な領域に DMA を行うことができます。DMA レジスタセットと代替マップレジスタセットとは別なものであるということは重要なことです。

OS の DMA レジスタセットの使用例は以下のとおりです。

1. DMA レジスタセットのアロケート
2. 現在のレジスタの取得
3. DMA レジスタセットの設定



4. 要求されたメモリ内へのマップ
5. DMA レジスタセットの取得
6. 本来のレジスタセットの設定
7. DMA レジスタセットへ希望する DMA チャンネルの割り当て

前述のコールにより、要求された DMA チャンネルへ全 DMA アクセスが行われる場合は、現在のレジスタセットに関係なく、現在の DMA レジスタセットを通じてマッピングできるようになります。言い換えれば、DMA レジスタセットは、指定の DMA チャンネル上の DMA オペレーションのために現在のマッピングレジスタを無視することになります。DMA レジスタセットに割り当てられていない DMA チャンネルは、そのすべての DMA を現在のマッピングレジスタセットを通してマッピングされます。

## INT 67H

## Get Alternate Map Register Set

ファンクション No. 28  
コード 00H

**機能** 代替マップレジスタセットの取得

**コール** AX = 5B00H

**リターン** AH = 00H 正常実行 (代替マップレジスタセットを取得した)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 83H 回復不可 (指定の EMM ハンドルがない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (サブファンクションパラメータが無効)  
 = A4H 回復不可 (OS がこのファンクションのアクセスを拒否している、  
 このときファンクションは使用できない)

**BL** カレントのアクティブな代替レジスタマップセットの番号

BL ≠ 0 の場合、ファンクションがコールされているときに使用されているマップレジスタセットを含む ES:DI は使用されない。

BL = 0 の場合、システム内の全マップレジスタの状態を含むエリアへのポインタ。また本来の状態に戻すために返すに必要な追加情報が返されたことを示す。

**ES:DI** マップレジスタコンテキストのセーブエリアへのポインタ

OS が供給しているコンテキストのセーブエリアへのポインタを含む。このポインタはセグメント:オフセットの形式を持つ。もし拡張メモリハードウェアが代替マッピングレジスタセットを供給していなければ、このポインタは常に返される。OS は Set Alternate Map Register Set サブファンクションをコールするときは、必ずこのポインタを拡張メモリマネージャに渡す。OS が Set Alternate Map Register Set サブファンクションをコールする前にこのファンクションをコールしたら、このファンクションはポインタ値を 0 にして返す。OS はセーブエリア用にスペースを確保しておかなくてはならない。しかし有効な情報が格納される前に、OS はメモリマネージャにこのセーブエリアの内容を初期化するように要求する必要がある。

OS は、ファンクション 15 (Get Page Map) によりアロケートされたセーブエリアを初期化する。初期化終了後、そのセーブエリアにはシステム内の全ボード

上のマップレジスタの状態が全部格納される。このセーブエリアには、OS が Set Alternate Map Register Set サブファンクションをコールするときに、本来の状態に戻すために必要な追加情報も格納されている。

# 解説

このファンクションは、このファンクションがコールされているときにアクティブになっているマップレジスタによって、以下の2つのうちいずれかを実行します。

1. Set Alternate Map Register Set サブファンクションが、代替マップレジスタセットに 0 (BL = 0) を返して終了していた場合、以下のようになります

a. Set Alternate Map Register Set サブファンクションによって EMM 内に保存されたコンテキスト保存領域ポインタは、この呼び出しによって返されます。このポインタは常に、代替マップレジスタセットを用意しないボードのために返されます。

b. 返されたコンテキスト保存領域ポインタがゼロでなければ、このサブファンクションは、システムの各拡張メモリボードのマップレジスタの内容を、このポインタが指定する保存領域にコピーします。この保存領域の書式は、ファンクション 15 (Get Page Map サブファンクション) によって返されるものと同じになります。これは代替マップレジスタセットを得ることをシミュレートする目的で行われます。メモリマネージャはこのマッピングコンテキストにスペースをアロケートしないことに注意してください。これはオペレーティングシステムが行います。

c. 返されたコンテキスト保存領域ポインタがゼロであれば、このサブファンクションは、システムの各拡張メモリボードのマップレジスタの内容をこのポインタが指定する保存領域にコピーすることはありません。

d. コンテキスト保存領域ポインタは、先行する Set Alternate Map Register Set ファンクションの呼び出しによって初期設定されていなければなりません。EMM 内に保存されているコンテキスト保存領域ポインタは、インストールの直後はゼロであることに注意してください。

e. コンテキスト保存領域は先行する Get Page Map ファンクション (ファンクション 15) の呼び出しによって初期設定されなければなりません。



2. Set Alternate Map Register Set サブファンクションのコールが、0 より大きい (BL > 0) 代替マップレジスタセットを返して終了した場合、このファンクションがコールされたときに、使用されている代替マップレジスタセットの番号が返されます。

|   |
|---|
| 例 |
|---|

```

alt_map_reg_set DB ?
context_save_area_ptr_seg DW ?
context_save_area_ptr_offset DW ?

MOV AX,5B00H ; load function code
INT 67H ; call the memory manager
OR AH,AH ; check EMM status
JNZ emm_err_handler ; jump to error handler on error
MOV alt_map_reg_set,BX ; save alt map reg set
TEST BL,BL ;
JNZ no_ptr_returned ;
MOV context_save_area_ptr_seg,ES ;
MOV context_save_area_ptr_offset,DI ;
no_ptr_returned:

```

## INT 67H

## Set Alternate Map Register Set

ファンクション No. 28  
コード 01H**機能** 代替マップレジスタセットのセット**コール** AX = 5B01H**BL** 新しい代替マップレジスタセットの番号

アクティブな代替マップレジスタセットの番号

BL≠0 の場合、マップレジスタコンテキストのリストアエリアへのポインタは要求されない。また ES:DI の内容は影響を受けず無視される。ボードがレジスタをサポートしていれば BL 内の指定された代替マップレジスタセットは、アクティブになる。

BL=0 の場合、システム内の全ボードの全マップレジスタの状態を含むエリアへのポインタと、本来の状態に戻すために必要な追加情報が、ES:DI に返されたことを示す。

**ES:DI** マッピングレジスタコンテキストのリストアエリアへのポインタ

OS が供給している、マップレジスタコンテキストのリストアエリアへのポインタを含む。このポインタはセグメント:オフセットの形をとり、拡張メモリハードウェアが代替マッピングレジスタセットを供給していなければ、常に返される。

OS がこのファンクションをコールするときは、いつでもこのポインタをメモリマネージャに引渡さなければならない。はリストアエリアのためにあらかじめスペースを確保しなくてはならない。さらに、このリストアエリアの内容は、有効な情報を格納する前にメモリマネージャが初期化するように、OS が要求を出さなくてはならない。

OS はファンクション 15 (Get Page Map) により確保したリストアエリアを初期化する。初期化終了後、このエリアはマップレジスタの状態を含むことになる。OS が Set Alternate Map Register Set サブファンクションをコールするとき、このリストアエリアは本来の状態に戻すために必要な追加情報をも含む。

**リターン** AH = 00H 正常実行 (代替マップレジスタをセットした)

= 80H 回復不可 (メモリマネージャソフトウェアが動作しない)

= 81H 回復不可 (拡張メモリハードウェアが動作しない)

= 83H 回復不可 (指定の EMM ハンドルがない)

- = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)
- = 8FH 回復不可 (サブファンクションパラメータが無効)
- = 9AH 回復不可 (代替マップレジスタセットはサポートされているが、指定の代替マップレジスタセットがサポートされない)
- = 9CH 回復不可 (代替マップレジスタセットはサポートされているが、指定の代替マップレジスタセットは0ではない)
- = 9DH 回復不可 (代替マップレジスタセットはサポートされているが、指定の代替マップレジスタセットが定義されていないか、アロケートされていない)
- = A3H 回復不可 (ソースアレイの内容が違うか、サブファンクションから渡されたポインタが無効である)
- = A4H 回復不可 (OSがこのファンクションのアクセスを拒否している。このときファンクションは使用できない)

# 解説

このファンクションは、指定のマップレジスタセットに従って、次の2つのうち1つを実行します。

1. もし設定された代替マップレジスタセットがゼロであれば、そのマップレジスタセットがアクティブになります。マップレジスタコンテキスト復元領域ポインタがゼロでなければ、ES:DIによって示された復元領域の内容を、システム内の各拡張メモリボードのレジスタセットにコピーします。このポインタがゼロであれば、その内容はコピーされません。

マップレジスタコンテキスト復元領域ポインタは、その値にかかわらず、メモリマネージャ内に保存されます。この値は、Get Alternate Map Register Set サブファンクションによって使用されます。

オペレーティングシステムは、このコンテキスト復元領域ポインタを用意しなければなりません。このサブファンクションは、代替マップレジスタセットの設定をシミュレートする目的で使われます。メモリマネージャはコンテキスト保存領域ポインタを内部的に保存し、このマッピングコンテキストにスペースをアロケートするのはオペレーティングシステムが行うことに注意してください。

2. もし指定の変更マップレジスタセットが0でなければ、指定のマップレジスタセットがアクティブです。OSがポイントしているリストアエリアは、使用されることはありません。



|                 |                                    |                                  |
|-----------------|------------------------------------|----------------------------------|
| 例               | alt_map_reg_set                    | DB ?                             |
|                 | context_restore_area_ptr_seg       | DW ?                             |
|                 | context_restore_area_ptr_offset    | DW ?                             |
|                 |                                    |                                  |
| MOV             | AX,5B01H                           | ; load function code             |
| MOV             | BL,alt_map_reg_set                 | ; set alt map reg set            |
| TEST            | BL,BL                              | ;                                |
| JZ              | no_ptr_passed                      | ;                                |
| MOV             | ES,context_restore_area_ptr_seg    | ;                                |
| MOV             | DI,context_restore_area_ptr_offset | ;                                |
|                 |                                    |                                  |
| no_ptr_passed : |                                    |                                  |
| INT             | 67H                                | ; call the memory manager        |
| OR              | AH,AH                              | ; check EMM status               |
| JNZ             | emm_err_handler                    | ; jump to error handler on error |

## INT 67H

Get Alternate Map Save Array Size

 ファンクション No. 28  
 コード 02H

機能 代替マップセーブ配列のサイズ取得

コール AX = 5B02H

リターン AH = 00H 正常実行 (配列の大きさを返した)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 83H 回復不可 (指定の EMM ハンドルがない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (パラメータが無効)  
 = A4H 回復不可 (OS がこのファンクションのアクセスを拒否している。  
 このときファンクションは使用できない)

## DX 配列のサイズ

OS が Get, Set, Get & Set のファンクションに要求するときに必ず、OS が供給しているメモリエリアに転送されるバイト数を含む。

解説 このファンクションは、他のファンクション 28 に参照されるマップレジスタコンテキストをセーブするために必要な大きさを返します。

例

size\_of\_array

DW ?

|     |                  |                                  |
|-----|------------------|----------------------------------|
| MOV | AX,5B02H         | ; load function code             |
| INT | 67H              | ; call the memory manager        |
| OR  | AH,AH            | ; check EMM status               |
| JNZ | emm_err_handler  | ; jump to error handler on error |
| MOV | size_of_array,DX | ; save size of array             |

## INT 67H

## Allocate Alternate Map Register Set

ファンクション No. 28  
コード 03H**機能** 代替マップレジスタセットのアロケート**コール** AX = 5B03H

**リターン** AH = 00H 正常実行 (代替マップレジスタの番号を返した)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 83H 回復不可 (指定の EMM ハンドルがない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (パラメータが無効)  
 = 9BH 回復不可 (代替マップレジスタはサポートされているが、現在代替マップレジスタはすべてアロケートされている)  
 = A4H 回復不可 (OS がこのファンクションのアクセスを拒否している。このときファンクションは使用できない)

**BL 代替マップレジスタセットの番号**

代替マップレジスタセットの番号を含む。もしハードウェアが代替マップレジスタセットをサポートしていなければ、0 が返される。この場合、マップレジスタコンテキストセーブエリアへのポインタを取得するために、ファンクション 28 がコールされなければならない。OS は、このセーブエリアをサポートしなくてはならない。このセーブエリアは、ハードウェアが変更マップレジスタセットをサポートしていないため、必要である。

**解説** このサブファンクションは、ファンクションがコールされたときに使用できる代替マップレジスタセットの番号を返します。もしハードウェアが代替マップレジスタをサポートしていなければ、代替マップレジスタセットの番号には 0 が返されます。

アロケートされた代替マップレジスタセットは、Get または Set Alternate Map Register Set ファンクションを使用するときに、この返された番号で参照されます。OS はこれらのサブファンクションを使い、代替マップレジスタを参照し、マッピングコンテキストを拡張メモリ上ですぐ切り換えることができます。



このファンクションは、アクティブな代替マップレジスタセットの内容を、新しくアロケートされた代替マップレジスタセットのマッピングレジスタにコピーします。これにより、OS が Set Alternate Map Register Set ファンクションを実行するときに、新しい代替マップのアロケートの前にマップされたメモリへの読み書きが可能になります。このファンクションは、代替マップレジスタセットを実際に使用中のマップレジスタセットを変更することはありませんが、新しい代替マップレジスタセットをアロケートすることに加えて、後続く Set Alternate Map Register Set のために新しい代替マップレジスタセットを用意します。

| 例 | alt_map_reg_num        | DB ?                             |
|---|------------------------|----------------------------------|
|   | MOV AX,5B03H           | ; load function code             |
|   | INT 67H                | ; call the memory manager        |
|   | OR AH,AH               | ; check EMM status               |
|   | JNZ emm_err_handler    | ; jump to error handler on error |
|   | MOV alt_map_reg_num,BL | ; save alt map reg num           |

## INT 67H

## Deallocate Alternate Map Register Set

ファンクション No. 28  
コード 04H

**機能** 代替マップレジスタセットの解放

**コール** AX = 5B04H

BL 代替マップレジスタセットの番号

解放される代替マップレジスタセットの番号。マップレジスタセットが0ならば、アロケートまたは解放はできない。しかし、もし変更マップレジスタセットに0が指定されており、本サブファンクションがコールされていた場合は、エラーにはならない。この場合、このファンクションのコールは無視される。

**リターン**

- AH = 00H 正常実行 (代替マップレジスタセットは解放された)
- = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)
- = 81H 回復不可 (拡張メモリハードウェアが動作しない)
- = 83H 回復不可 (指定の EMM ハンドルがない)
- = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)
- = 8FH 回復不可 (パラメータが無効)
- = 9CH 回復不可 (代替マップレジスタセットはサポートされているが、指定の代替マップレジスタセットが0ではない)
- = 9DH 回復不可 (代替マップレジスタセットはサポートされているが、指定の代替マップレジスタセットが定義されていないか、アロケートされていない)
- = A4H 回復不可 (OS がこのファンクションのアクセスを拒否している。このときファンクションは使用できない)

**解説** Deallocate Alternate Map Register Set サブファンクションは、代替マップレジスタセットを、将来の使用のためにメモリマネージャに返します。メモリマネージャは必要なときに、この代替マップレジスタを再アロケートできます。

このファンクションは、指定の代替マップレジスタのマッピングコンテキストを読み書き不可能 (解除) にもできます。これにより、以前にこの代替マップレジスタにマッピングされたページをアクセス不可にして保護します。現在の代替マップレジスタセットは解放はできませんので注意してください。これを行うと、現在の内部メモリと拡張メモリにマッピングされているすべてのメモリが、アクセス不可になります。

|     |                          |                                  |
|-----|--------------------------|----------------------------------|
| 例   | alternate_map_reg_set    | DB ?                             |
| MOV | BX,alternate_map_reg_set | ; set alternate map reg set      |
| MOV | AX,5B04H                 | ; load function code             |
| INT | 67H                      | ; call the memory manager        |
| OR  | AH,AH                    | ; check EMM status               |
| JNZ | emm_err_handler          | ; jump to error handler on error |



## INT 67H

## Allocate DMA Register Set

ファンクション No. 28  
コード 05H

**機能** DMA レジスタセットのアロケート

**コール** AX = 5B05H

**リターン** AH = 00H 正常実行 (DMA レジスタセットをアロケートした)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 83H 回復不可 (指定の EMM ハンドルがない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (パラメータが無効)  
 = 9BH 回復不可 (DMA レジスタセットはサポートされているが、現在  
 DMA レジスタセットは全部アロケートされている)  
 = A4H 回復不可 (OS がこのファンクションのアクセスを拒否している。  
 このときファンクションは使用できない)

**BL DMA レジスタセットの番号**

DMA レジスタセットの番号を含む。もし DMA レジスタセットがハードウェアによりサポートされていない場合は 0 が返る。

注) PC-9800シリーズでは、DMAレジスタセットはサポートしていません。

**解説** Allocate DMA Register Set サブファンクションは、もし DMA レジスタセットが現在使用可能ならば、DMA レジスタセットの番号を返します。もしハードウェアが DMA レジスタセットをサポートしていなければ、DMA レジスタセット番号には 0 が返されます。

マルチタスク OS では、あるタスクが DMA が完了するのを待っている間、別なタスクに切り換えができると便利です。しかし、DMA が現在のレジスタセットを通じてマップされている間は、この切り換えはできません。つまり全 DMA 動作は、ページの再マップの前に、完了しておかなければなりません。

OS は、指定の代替マップレジスタセットを使用して、指定の DMA チャンネルでの DMA オペレーションを開始するでしょう。この代替マップレジスタセットは、DMA オペレーションが完了するまで、OS またはアプリケーションプログラ

ムにより、再度使用されることはありません。OSは代替マップレジスタの内容を変更しないことと、DMAオペレーションの間アプリケーションプログラムが代替マップレジスタセットの内容を変更させないようにすることにより、DMAの動作を保証します。

| 例   | DMA_reg_set_number    | DB ?                             |
|-----|-----------------------|----------------------------------|
| MOV | AX,5B04H              | ; load function code             |
| INT | 67H                   | ; call the memory manager        |
| OR  | AH,AH                 | ; check EMM status               |
| JNZ | emm_err_handler       | ; jump to error handler on error |
| MOV | DMA_reg_set_number,BL | ; save DMA reg set number        |

## INT 67H

Enable DMA on Alternate Map Register Set

ファンクション No. 28

コード 06H

機能 代替マップレジスタ上の DMA の使用許可

コール AX = 5B06H

BL DMA レジスタセット番号

DL で指定される DMA チャンネル上の、DMA オペレーションに使用される代替マップレジスタセットの番号。もし指定の代替マップレジスタセットが 0 ならば、指定の DMA チャンネルに対する DMA アクセスは実行されない。

DL DMA チャンネル番号

BL で指定された DMA マップレジスタセットに対応する DMA チャンネル。

リターン AH = 00H 正常実行 (代替マップレジスタの DMA の使用を許可した)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 83H 回復不可 (指定の EMM ハンドルがない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (パラメータが無効)  
 = 9AH 回復不可 (DMA レジスタセットはサポートされているが、指定の代替マップレジスタセットはサポートされていない)  
 = 9CH 回復不可 (DMA レジスタセットはサポートされていない。また指定の DMA レジスタセットは 0 でない)  
 = 9DH 回復不可 (DMA レジスタセットはサポートされているが指定の DMA レジスタセットは定義されていないか、アロケートされていない)  
 = 9EH 回復不可 (DMA チャンネルはサポートされていない)  
 = 9FH 回復不可 (DMA チャンネルはサポートされているが、指定の DMA チャンネルはサポートされていない)  
 = A4H 回復不可 (OS がこのファンクションのアクセスを拒否している。このときファンクションは使用できない)

注) PC-9800シリーズではDMAレジスタセットをサポートしていません。



**解 説**

このファンクションは、指定の代替マップレジスタセットを通じて指定の DMA チャンネルでの DMA アクセスを可能にします。マルチタスクの OS では、あるタスクが DMA が完了するのを待っている間、別なタスクに切り換えができると便利です。指定のチャンネルでの DMA は、現在のレジスタセットと関係なく指定の DMA レジスタセットを利用できます。もし、DMA チャンネルが DMA レジスタセットに割り付けられていなければ、そのチャンネルに対する DMA は現在のレジスタセットを通じてマップされます。

**例**

|                 |                                                  |
|-----------------|--------------------------------------------------|
| alt_map_reg_set | DB ?                                             |
| DMA_channel_num | DB ?                                             |
| MOV             | BL,alt_map_reg_set ; set alt map reg set         |
| MOV             | DL,DMA_channel_num ; set DMA channel num         |
| MOV             | AX,5B06H ; load function code                    |
| INT             | 67H ; call the memory manager                    |
| OR              | AH,AH ; check EMM status                         |
| JNZ             | emm_err_handler ; jump to error handler on error |

## INT 67H

Disable DMA on Alternate Map Register Set

ファンクション No. 28  
コード 07H**機能** 代替マップレジスタ上の DMA の使用不許可**コール** AX = 5B07H

BL 代替マップレジスタセットの番号

オペレーションを無効にする DMA レジスタセットの番号。もし指定のマップレジスタセットが 0 ならば、何も処理されない。

**リターン** AH = 00H 正常実行 (代替マップレジスタの DMA を使用不可にした)

= 80H 回復不可 (メモリマネージャソフトウェアが動作しない)

= 81H 回復不可 (拡張メモリハードウェアが動作しない)

= 83H 回復不可 (指定の EMM ハンドルがない)

= 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)

= 8FH 回復不可 (パラメータが無効)

= 9AH 回復不可 (DMA レジスタセットはサポートされているが、指定の代替マップレジスタセットはサポートされていない)

= 9CH 回復不可 (DMA レジスタセットはサポートされていない。また指定の DMA レジスタセットは 0 でない)

= 9DH 回復不可 (DMA レジスタセットはサポートされているが指定の DMA レジスタセットは定義されていないか、アロケートされていない)

= 9EH 回復不可 (DMA チャンネルはサポートされていない)

= 9FH 回復不可 (DMA チャンネルはサポートされているが、指定の DMA チャンネルはサポートされていない)

= A4H 回復不可 (OS がこのファンクションのアクセスを拒否している。このときファンクションは使用できない)

注) PC-9800 シリーズでは DMA レジスタセットをサポートしていません。

**解説** このファンクションは、指定の代替マップレジスタセットに対応する全 DMA チャンネルへの DMA アクセスを不可能にします。

例

DMA\_reg\_set

DB ?

MOV BL,DMA\_reg\_set

; set DMA reg set

MOV AX,5B07H

; load function code

INT 67H

; call the memory manager

OR AH,AH

; check EMM status

JNZ emm\_err\_handler

; jump to error handler on error



## INT 67H

## Deallocate DMA Register Set

ファンクション No. 28  
コード 08H**機能** DMA レジスタセットの解放**コール** AX = 5B08H

**リターン** AH = 00H 正常実行 (DMA レジスタセットは解放された)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 83H 回復不可 (指定の EMM ハンドルがない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (パラメータが無効)  
 = 9CH 回復不可 (DMA レジスタセットはサポートされていない。また指定の DMA レジスタセットは 0 でない)  
 = 9DH 回復不可 (DMA レジスタセットはサポートされているが指定の DMA レジスタセットは定義されていないか、アロケートされていない)  
 = A4H 回復不可 (OS がこのファンクションのアクセスを拒否している。このときファンクションは使用できない)

注) PC-9800 シリーズでは DMA レジスタセットをサポートしていません。

**BL DMA レジスタセットの番号**

DMA オペレーションで使用しなくなる DMA レジスタセットの番号。これは以前に指定の DMA チャンネルでの DMA のためにアロケートされ、使用を許可されていたレジスタセットについて行う。もし指定の DMA レジスタセットが 0 ならば、何も処理されない。

**解説** このファンクションは、指定の DMA レジスタセットを解放します。

|          |                    |                                  |
|----------|--------------------|----------------------------------|
| <b>例</b> | DMA_reg_set_num    | DB ?                             |
| MOV      | AX,5B08H           | ; load function code             |
| INT      | 67H                | ; call the memory manager        |
| OR       | AH,AH              | ; check EMM status               |
| JNZ      | emm_err_handler    | ; jump to error handler on error |
| MOV      | DMA_reg_set_num,BL | ; save DMA reg set num           |

## INT 67H

|                                                |                |
|------------------------------------------------|----------------|
| Prepare Expanded Memory Hardware For Warm Boot | ファンクション No. 29 |
|------------------------------------------------|----------------|

- 機能** ウォームブートのために拡張メモリハードウェアを準備する
- コール** AH = 5CH
- リターン**
- AH = 00H 正常実行 (ハードウェアの準備ができた)
  - = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)
  - = 81H 回復不可 (拡張メモリハードウェアが動作しない)
  - = 83H 回復不可 (指定の EMM ハンドルがない)
  - = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)

注) PC-9800 シリーズでは常に AH に 0 が返されます。

**解説** このファンクションは、緊急のウォームブートのために拡張メモリハードウェアを準備します。このファンクションは、OS が次にする動作はシステムのウォームブートであるとみなします。一般にファンクションは、現在のマッピングコンテキストや、使用中の代替マップレジスタセット、またウォームブート時に初期化の必要がある拡張メモリハードウェアに依存しているものすべてに影響を及ぼします。もし、アプリケーションプログラムが 640K バイト以下のアドレスにメモリをマップしようとするれば、アプリケーションプログラムはウォームブートに続くすべての可能な状況をトラップし、アプリケーションプログラム自体をウォームブートする前にこのファンクションをコールしなくてはなりません。

|   |     |                 |                                  |
|---|-----|-----------------|----------------------------------|
| 例 | MOV | AH, 5CH         | ; load function code             |
|   | INT | 67H             | ; call the memory manager        |
|   | OR  | AH, AH          | ; check EMM status               |
|   | JNZ | emm_err_handler | ; jump to error handler on error |



## INT 67H

## Enable OS/E Function Set

ファンクション No. 30  
コード 00H

**機能** OS ファンクションセットの使用許可

**コール** AX = 5D00H

BX, CX アクセスキー

2回目以降の、全ファンクションコールで必要とされる。アクセスキーは、最初のファンクションコールで返された値が要求される。

**リターン** AH = 00H 正常実行 (OS ファンクションが使用可能になった)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (パラメータが無効)  
 = A4H 回復不可 (OS が、このファンクションのアクセスを拒否している。  
 この場合、このファンクションは使用できない。このファンクションにより渡されるキーの値はこのファンクションを実行を許可するものではない)

BX, CX アクセスキー

最初のファンクションコール時に返される。メモリマネージャは、以降このファンクションを実行するために必要な、ランダム値のキーを返す。2回目以降のファンクションコールでは、このキーは返されない。このファンクションが2回目以降にコールされた場合は、BX, CX は影響を受けない。

**解説** このファンクションは、OS 指定のファンクションを使用できるように、OS が全プログラムまたはデバイスドライバに使用許可を与えるものです。この機能は、マップ可能な内部メモリの領域を管理する OS のためにのみ用意されており、プログラムが、マップ可能な内部メモリ領域に影響を与えるようなファンクションを使用することは許可されていません。しかしこの機能のためには、このファンクションを使用することはできます。OS がこのファンクションを使用禁止にしているとき、プログラムがこのファンクションを使用しようとする、メモリマネージャは OS がファンクションへのアクセスを拒否しているというステータスを返します。これは使用禁止状態のとき、ファンクションは動作しないというこ



とです。しかし使用許可状態のときは、すべてのプログラムはファンクションを使用することが可能です。

サブファンクションにより使用可能となる、OS/E (Operating System/Environment)ファンクションは、以下のものです。

1. ファンクション 26 : Get Expanded Memory Hardware Information
2. ファンクション 28 : Alternate Map Register Set
3. ファンクション 30 : Enable/Disable Operating System

これらのファンクションを再度使用可能にするファンクション自体が使用を禁止されているとき、OS がこれらのファンクションの再使用許可ができるということは矛盾しているように見えます。その手順の概要について以下に説明します。

メモリマネージャはロードされたとき、このファンクションも含めて、OS 指定の全ファンクションを使用可にします。OS は、他のソフトウェアがコールする前に、Enable/Disable OS/E Function Set サブファンクションをコールすることにより、これらのファンクションへの限られたアクセスを取得します。

このサブファンクションのうちどれか1つを最初にコールしたとき、メモリマネージャは、OS がこれらのサブファンクションのどれかを将来コールするときに使用しなければならないアクセスキーを返します。メモリマネージャは、Enable/Disable OS/E Function Set サブファンクションを最初にコールするときには、アクセスキーは必要ありません。

以降、サブファンクションをコールするときには、このアクセスキーは Enable/Disable OS/E Function Set サブファンクションで必要となります。アクセスキーは Enable/Disable OS/E Function Set サブファンクションの最初のコールにのみ返されます。また、OS はこのファンクションをコールする最初のソフトウェアであるはずですから、OS のみがこのキーのコピーを得ることができます。メモリマネージャはランダム値でアクセスキーを返さなくてはなりません。固定値のキーを返すと OS への安全保護の目的が失われます。

**例**

## 1. 最初のコール

access\_key

DW 2 DUP (?)

```

MOV AX,5D00H ; load function code
INT 67H ; call the memory manager
OR AH,AH ; check EMM status
JNZ emm_err_handler ; jump to error handler on error
MOV access_key [0],BX ; save access key
MOV access_key [2],CX ;

```

## 2. 2 回目以降のコール

access\_key

DW 2 DUP (?)

```

MOV BX,access_key [0] ; set access key
MOV CX,access_key [2] ;
MOV AX,5D00H ; load function code
INT 67H ; call the memory manager
OR AH,AH ; check EMM status
JNZ emm_err_hendler ; jump to error handler on error

```

**注 意**

このファンクションは、OS によってのみ使用されます。OS は、いつでもこのファンクションを使用不可にできます。

## INT 67H

## Disable OS/E Function Set

ファンクション No. 30  
コード 01H

**機能** OS/E ファンクションセットの使用禁止状態の設定

**コール** AX = 5D01H

BX, CX アクセスキー

(2回目以降の全ファンクションコールで必要とされる。アクセスキーは、最初のファンクションコールで返された値が要求される)

**リターン** AH = 00H 正常実行 (OS/E ファンクションを使用禁止にした)  
 = 80H 回復不可 (メモリマネージャソフトウェアが動作しない)  
 = 81H 回復不可 (拡張メモリハードウェアが動作しない)  
 = 84H 回復不可 (メモリマネージャに渡されたファンクションが未定義)  
 = 8FH 回復不可 (パラメータが無効)  
 = A4H 回復不可 (OSがこのファンクションへのアクセスを拒否している。この場合、このファンクションは使用できない。このファンクションにより渡されたキーの値はこのファンクションの実行を許可するものではない)

BX, CX アクセスキー

最初のファンクションコール時のみ返される。メモリマネージャは、以降このファンクションを実行するために必要なランダム値のキーを返す。2回目以降のファンクションコールでは、このキーは返されない。このファンクションが2回目以降にコールされた場合は、BX, CX は影響を受けない。

**解説** このファンクションは、OSが全プログラムまたはデバイスドライバがOS指定のファンクションを使用できないようにします。この機能は、マップ可能な内部メモリの領域を管理するOSのためにのみ用意され、プログラムがマップ可能な内部メモリ領域に影響を与えるようなファンクションを使用することはできません。するとこの機能はこれらのファンクションを使用できなくなってしまう。OSがこのファンクションを使用禁止にしているとき、プログラムがこのファンクションを使用しようとする、メモリマネージャはOSがファンクションへのアクセスを否定しているというステータスを返します。言い換えれば使用状態のときは、ファンクションは動作しないということです。



OS/E (Operating System/Environment) ファンクションのサブファンクションは、以下の動作が可能です。

1. ファンクション 26 : Get Expanded Memory Hardware Information
2. ファンクション 28 : Alternate Map Register Set
3. ファンクション 30 : Enable/Disable Operating System

|   |
|---|
| 例 |
|---|

## 1. 最初のコール

|                       |                                  |
|-----------------------|----------------------------------|
| access_key            | DW 2 DUP (?)                     |
| MOV AX,5D01H          | ; load function code             |
| INT 67H               | ; call the memory manager        |
| OR AH,AH              | ; check EMM status               |
| JNZ emm_err_hendler   | ; jump to error handler on error |
| MOV access_key [0],BX | ; save access key                |
| MOV access_key [2],CX | ;                                |

## 2. 2 回目以降のコール

|                       |                                  |
|-----------------------|----------------------------------|
| access_key            | DW 2 DUP (?)                     |
| MOV BX,access_key [0] | ; set access key                 |
| MOV CX,access_key [2] | ;                                |
| MOV AX,5D01H          | ; load function code             |
| INT 67H               | ; call the memory manager        |
| OR AH,AH              | ; check EMM status               |
| JNZ emm_err_hendler   | ; jump to error handler on error |

|     |
|-----|
| 注 意 |
|-----|

このファンクションは、OS によってのみ使用されます。OS は、いつでもこのファンクションを使用禁止にできます。

## INT 67H

## Return Access Key

ファンクション No. 30  
コード 02H

**機能**      アクセスキーのリターン

**コール**      AX = 5D02H  
              BX, CX    アクセスキー

2回目以降の全ファンクションコールで必要とされる。アクセスキーは、最初のファンクションコールで返された値が要求される。

**リターン**    AH = 00H    正常実行 (アクセスキーがメモリマネージャに返された)  
              = 80H    回復不可 (メモリマネージャソフトウェアが動作しない)  
              = 81H    回復不可 (拡張メモリハードウェアが動作しない)  
              = 84H    回復不可 (メモリマネージャに渡されたファンクションが未定義)  
              = 8FH    回復不可 (パラメータが無効)  
              = A4H    回復不可 (OSがこのファンクションへのアクセスを拒否している。この場合、このファンクションは使用できない。このファンクションに渡されたキーの値はこのファンクションを実行を許可するものではない)

**解説**      このファンクションは、OSがメモリマネージャにアクセスキーを返せるようにする機能を持ちます。メモリマネージャにアクセスキーを返すと、メモリマネージャはインストール時の状態 (OS/E ファンクションセットとアクセスキーの使用に関して) になります。つまり OS/E ファンクションセットへのアクセスが使用許可になるわけです。次回の Enable/Disable OS/E Function Set のサブファンクションの実行により、アクセスキーが再び与えられます。

|          |                          |                                  |
|----------|--------------------------|----------------------------------|
| <b>例</b> | access_key               | DW 2 DUP (?)                     |
|          | MOV    BX,access_key [0] | ; set access key                 |
|          | MOV    CX,access_key [2] | ;                                |
|          | MOV    AX,5D02H          | ; load function code             |
|          | INT    67H               | ; call the memory manager        |
|          | OR     AH,AH             | ; check EMM status               |
|          | JNZ    emm_err_hendler   | ; jump to error handler on error |

**注 意**

このファンクションは OS によってのみ使用されます。OS はいつでもこのファンクションを使用禁止にすることができます。



## INT 67H

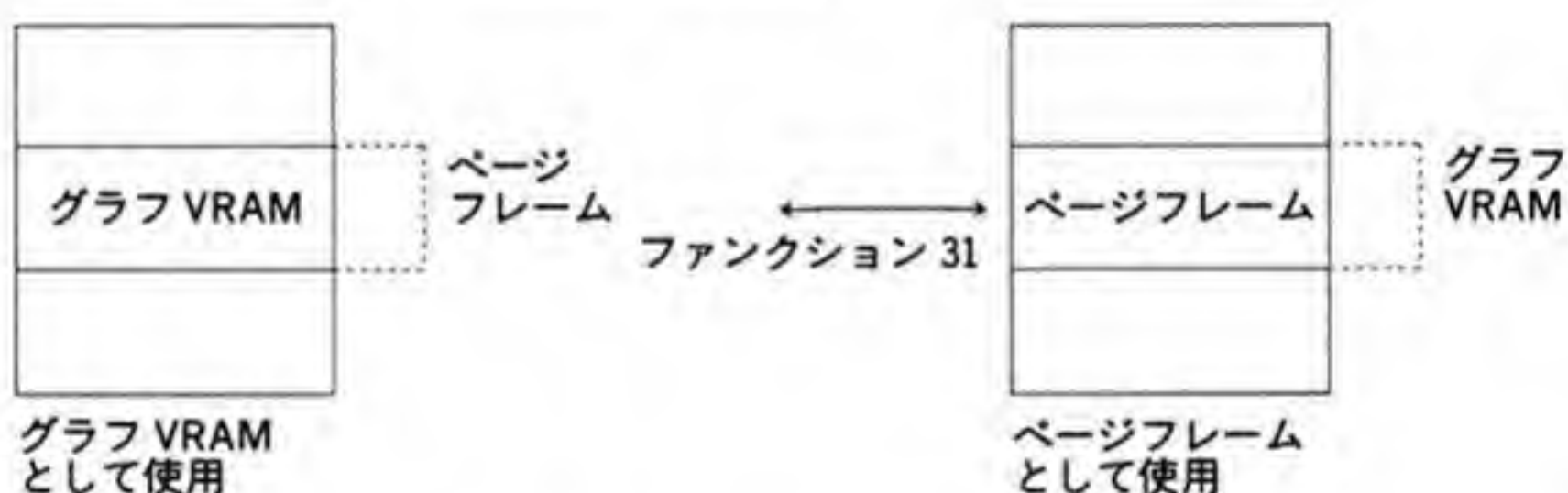
## ファンクション No. 31

機能 ページフレームの管理

注意 PC-9800 シリーズでは、ページフレームとして使用するメモリアドレスが、グラフィック VRAM の一部 (B0000H~BFFFFH) と重なっており、切り換えによってどちらでも使用できる機種 (注1) があります。

このような機種では、そのメモリとページフレームとして使用する場合、グラフィック VRAM として使用する場合でアプリケーションプログラムがメモリを切替えなければなりません。

このファンクション 31 は、グラフィック VRAM とページフレームの切替え (または状態取得) のために提供されます。



ファンクション 31 には、以下のようなサブファンクションが用意されています。

1. ページフレーム用バンクのステータス取得
2. ページフレーム用バンクの切替え

EMSドライバは、次に示すファンクションを実行する場合はページフレーム用バンクに切り換えてしまうため、グラフィック VRAM を使用するアプリケーションプログラムは、必要に応じて VRAM バンクに切り換えてください。

## ファンクション

|    |                           |
|----|---------------------------|
| 5  | ページのマッピング                 |
| 8  | ページマップのセーブ                |
| 9  | ページマップのリストア               |
| 15 | ページマップのセーブ/リストア           |
| 16 | ページマップの一部セーブ/リストア         |
| 17 | 複数ページのマッピング               |
| 22 | ページの変更とジャンプ               |
| 23 | ページマップの変更とコール             |
| 24 | メモリ領域の移動                  |
| 28 | 代替マップレジスタによるページマップの取得/セット |

## ●ページフレーム用バンクの切り換え方法

ページフレーム用バンクがグラフィック VRAM の一部と重なっている機種において、EMS を使用するアプリケーションプログラムは、ページフレーム用バンクの切り換え操作を、次のような手順で行わなければなりません。



①アプリケーションプログラムの起動時に、ページフレーム用バンクのステータスを取得(ファンクション 31, コード 00H)して、セーブしておいてください。

②ページフレーム用バンクをページフレームとして使用する場合、ページフレーム用バンクをページフレームに切り換えてください(ファンクション 31, コード 01H)。ただし、ページフレーム用バンクがすでにページフレームに切り換わっている場合は、その必要はありません。

③EMS 関連の主処理を行います(ページのアロケート、ページのマッピング、ページフレームのアクセス等)。

④アプリケーションプログラムの終了時に、ページフレーム用バンクのステータスを①でセーブしていた内容に戻してください(ファンクション 31, コード 01H)。

- (注1) PC-9801RA, RX, LS, ES, EX, RS, LX, RL (ノーマルモード)  
(EMS機能のある機種)

その他の機種(ノーマルモード)では、ページフレームは C0000H または C8000H~CFFFFH となっています。

ハイレゾリレーションモードの機種では、ページフレームは B0000H ~BFFFFH となっていますが、グラフィック VRAM は C0000H~DFFFFFFH となっています。

これらの機種で、このファンクションを実行しても差しつかえありません。



## INT 67H

**Get Page frame Status****ファンクション No. 31**  
**コード 00H****機能** ページフレーム用バンクのステータスを取得する**コール** AX=7000H**リターン** AH=00H 正常実行 (ステータスを取得した)  
=80H 回復不可 (メモリマネージャソフトが動作しない)

AL ステータス

00H のとき、ページフレーム用バンクはページフレームとして使用可能、

01H のとき、ページフレーム用バンクはページフレームとして使用不可、

**解説** このファンクションは、ページフレーム用バンク (ページフレーム) の状態を返します。返された値によって現在ページフレーム用バンクのメモリをページフレームとして使用できるかどうかを判別できます。

## INT 67H

**Enable/Disable Page frame**

**ファンクション No. 31**  
**コード 01H**

**機能** ページフレーム用バンクの状態を設定

**コール** AX=7001H

BL 指示

00H のとき、ページフレーム用バンクをページフレームとして使用可能にする。

01H のとき、ページフレーム用バンクを VRAM として使用する。

**リターン** AH=00H 正常実行（ファンクションは正しく実行された）

80H 回復不可（メモリマネージャソフトウェアが動作しない）

AL ステータス

00H のとき、ページフレーム用バンクはページフレームとして使用可能

01H のとき、ページフレーム用バンクはページフレームとして使用不可

**解説** このファンクションは、指定された状態にページフレーム用バンクの切り換えを行います。OS、アプリケーションプログラムの開発者は、ファンクション 31 のこの 2 つのサブファンクションでページフレーム（ページフレーム用バンク）を管理しなければなりません。

## 5.5 ファンクションとステータスコードのクロスリファレンス

ここでは、2つのクロスリファレンスを掲載しています。1つは、ファンクションコードと、それが返すステータスコードとの対応を示す一覧表です。もう1つは、ステータスコードと、それらを返すファンクションとの関係を示す一覧表です。

表1 ファンクションとステータスコードのクロスリファレンス

| ファンクション | ステータス (16進)                        | 説明                                                            |
|---------|------------------------------------|---------------------------------------------------------------|
| 40H     | 00H,80H,81H,84H                    | Get memory manager status                                     |
| 41H     | 00H,80H,81H,84H                    | Get Page Frame Segment Address                                |
| 42H     | 00H,80H,81H,84H                    | Get Unallocated Page Count                                    |
| 43H     | 00H,80H,81H,84H,85H,87H,88H        | Allocate Pages                                                |
| 44H     | 00H,80H,81H,83H,84H,8AH,8BH        | Map/Unmap Handle Page                                         |
| 45H     | 00H,80H,81H,83H,84H,86H            | Deallocate Pages                                              |
| 46H     | 00H,80H,81H,84H                    | Get EMM Version                                               |
| 47H     | 00H,80H,81H,83H,84H,8CH,8DH        | Save Page Map                                                 |
| 48H     | 00H,80H,81H,83H,84H,8EH            | Restore Page Map                                              |
| 49H     |                                    | Reserved                                                      |
| 4AH     |                                    | Reserved                                                      |
| 4BH     | 00H,80H,81H,84H                    | Get EMM Handle Count                                          |
| 4CH     | 00H,80H,81H,83H,84H                | Get EMM Handle Pages                                          |
| 4DH     | 00H,80H,81H,84H                    | Get All EMM Handle Pages                                      |
| 4E00H   | 00H,80H,81H,84H,8FH                | Get Page Map                                                  |
| 4E01H   | 00H,80H,81H,84H,8FH,A3H            | Set Page Map                                                  |
| 4E02H   | 00H,80H,81H,84H,8FH,A3H            | Get & Set Page Map                                            |
| 4E03H   | 00H,80H,81H,84H,8FH                | Get Size of Page Map Save Array                               |
| 4F00H   | 00H,80H,81H,84H,8BH,8FH,A3H        | Get Partial Page Map                                          |
| 4F01H   | 00H,80H,81H,84H,8FH,A3H            | Set Partial Page Map                                          |
| 4F02H   | 00H,80H,81H,84H,8BH,8FH            | Get Size of Partial Page Map Save Array                       |
| 5000H   | 00H,80H,81H,83H,84H,8AH,8BH<br>8FH | Map/Unmap Multiple Handle Pages<br>(Physical pagenumber mode) |
| 5001H   | 00H,80H,81H,83H,84H,8AH,8BH<br>8FH | Map/Unmap Multiple Handle Pages<br>(Segment address mode)     |
| 51H     | 00H,80H,81H,83H,84H,87H,88H        | Reallocat Pages                                               |
| 5200H   | 00H,80H,81H,83H,84H,8FH,91H        | Get Handle Attribute                                          |



|       |                                                            |                                                 |
|-------|------------------------------------------------------------|-------------------------------------------------|
| 5201H | 00H,80H,81H,83H,84H,8FH,90H<br>91H                         | Set Handle Attribute                            |
| 5202H | 00H,80H,81H,84H,8FH                                        | Get Handle Attribute Capability                 |
| 5300H | 00H,80H,81H,83H,84H,8FH                                    | Get Handle Name                                 |
| 5301H | 00H,80H,81H,83H,84H,8FH,A1H                                | Set Handle Name                                 |
| 5400H | 00H,80H,81H,84H,8FH                                        | Get Handle Directory                            |
| 5401H | 00H,80H,81H,84H,8FH,A0H,A1H                                | Search for Named Handle                         |
| 5402H | 00H,80H,81H,84H,8FH                                        | Get Total Handles                               |
| 5500H | 00H,80H,81H,83H,84H,8AH,8BH<br>8FH                         | Alter Page Map & Jump<br>(Physical page mode)   |
| 5501H | 00H,80H,81H,83H,84H,8AH,8BH<br>8FH                         | Alter Page Map & Jump<br>(Segment address mode) |
| 5600H | 00H,80H,81H,83H,84H,8AH,8BH<br>8FH                         | Alter Page Map & Call<br>(Physical page mode)   |
| 5501H | 00H,80H,81H,83H,84H,8AH,8BH<br>8FH                         | Alter Page Map & Call<br>(Segment address mode) |
| 5602H | 00H,80H,81H,84H,8FH                                        | Get Alter Page Map & Call Stack Space Size      |
| 5700H | 00H,80H,81H,83H,84H,8AH,8FH<br>92H,93H,94H,95H,96H,98H,A2H | Move Memory Region                              |
| 5701H | 00H,80H,81H,83H,84H,8AH,8FH<br>93H,94H,95H,96H,97H,98H,A2H | Exchange Memory Region                          |
| 5800H | 00H,80H,81H,84H,8FH                                        | Get Mappable Physical Address Array             |
| 5801H | 00H,80H,81H,84H,8FH                                        | Get Mappable Physical Address Array Entries     |
| 5900H | 00H,80H,81H,84H,8FH,A4H                                    | Get Expanded Memory Hardware Information        |
| 5901H | 00H,80H,81H,84H,8FH                                        | Get Unallocated Raw Page Count                  |
| 5A00H | 00H,80H,81H,84H,85H,87H,88H<br>8FH                         | Allocate Standard pages                         |
| 5A01H | 00H,80H,81H,84H,85H,87H,88H<br>8FH                         | Allocate Raw Pages                              |
| 5B00H | 00H,80H,81H,84H,8FH,A4H                                    | Get Alternate Map Register Set                  |
| 5B01H | 00H,80H,81H,84H,8FH,9AH,9CH<br>9DH,A3H,A4H                 | Set Alternate Map Register Set                  |
| 5B02H | 00H,80H,81H,84H,8FH,A4H                                    | Get Alternate Map Save Array Size               |
| 5B03H | 00H,80H,81H,84H,8FH,9BH,A4H                                | Allocate Alternate Map Register Set             |
| 5B04H | 00H,80H,81H,84H,8FH,9CH,9DH                                | Deallocate Alternate Map Pregister Set          |

A4H

|       |                                                |                                           |
|-------|------------------------------------------------|-------------------------------------------|
| 5B05H | 00H,80H,81H,84H,8FH,9BH,A4H                    | Allocate DMA Register Set                 |
| 5B06H | 00h,80H,81H,84H,8FH,9AH,9CH<br>9DH,9EH,9FH,A4H | Enable DMA on Alternate Map Register Set  |
| 5B07H | 00H,80H,81H,84H,8FH,9AH,9CH<br>9DH,9EH,9FH,A4H | Disable DMA on Alternate Map Register Set |
| 5E08H | 00H,80H,81H,84H,8FH,9CH,9DH<br>A4H             | Deallocate DMA Register Set               |

|       |                         |                                               |
|-------|-------------------------|-----------------------------------------------|
| 5CH   | 00H,80H,81H,84H         | Prepare Expanded Memory Hardware for Warmboot |
| 5D00H | 00H,80H,81H,84H,8FH,A4H | Enable Operating Function Set                 |
| 5D01H | 00H,80H,81H,84H,8FH,A4H | Disable Operating System Function Set         |
| 5D02H | 00H,80H,81H,84H,8FH,A4H | Return Operating System Access Key            |



表2 ステータスとファンクションコードのクロスリファレンス

| ステータス | ファンクション                                                                                                          | 説明                                                                                                                                                                                                                  |
|-------|------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 00H   | 全部                                                                                                               | ファンクションは正常に終了した。                                                                                                                                                                                                    |
| 80H   | 全部                                                                                                               | メモリマネージャが、拡張メモリソフトウェアの障害を見つけた。もし、メモリマネージャが正常に操作されていれば起こらなかった状況が発見された。                                                                                                                                               |
| 81H   | 全部                                                                                                               | メモリマネージャが、拡張メモリハードウェアの障害を見つけた。もしメモリハードウェアが正常に動作していれば起こらなかった状況が発見された。その問題の原因を調べるために拡張メモリシステムまで診断するべきである。                                                                                                             |
| 82H   | なし                                                                                                               | このエラーコードは、メモリマネージャのバージョン3.2以上のバージョンでは返されない。メモリマネージャの初期バージョンでは、このコードは“busy”ステータスを意味していた。このステータスは、現在の要求が発生したときにすでに拡張メモリ要求を処理中で、他の要求を処理できないことを示している。メモリマネージャの3.2以上のバージョンでは、メモリマネージャは決して“busy”にはならず、いつでも要求を受け入れることができる。 |
| 83H   | 44H, 45H, 47H, 48H, 4CH, 5000H, 5001H, 51H, 5200H, 5201H, 5300H, 5301H, 5500H, 5501H, 5600H, 5601H, 5701H, 5701H | メモリマネージャが、指定したハンドルを見つけることができない。プログラムが指定したハンドルを壊したかもしれない。メモリマネージャは、指定されたハンドあるに関係があるどんな情報も持っていない。                                                                                                                     |
| 84H   | 全部                                                                                                               | マネージャに渡されたファンクションコードは、現在定義されていない。現在、定義されているファンクションは、40Hから5DHと、70Hである。                                                                                                                                               |
| 85H   | 43H, 5A00H, 5A01H                                                                                                | 現在、利用できるハンドルが存在しない。現在、すべての割り当て可能なハンドルが使用されている。プログラムは、他のプログラムがハンドルを解放することを待って、ハンドルの割り当ての再要求をすることができる。ハンドルは、255個までサポートできる。                                                                                            |



|     |                                                             |                                                                                                                                                              |
|-----|-------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 86H | 45H                                                         | マッピングコンテキスト復元エラーが発見された。このエラーは、プログラムがハンドルを返すことを試み、指示されたハンドルに対するコンテキストスタックに“マッピングコンテキスト”がまだあるときに生じる。プログラムは、ハンドルを返す前に、マッピングコンテキストを復元することによって、このエラーから回復することができる。 |
| 87H | 43H, 51H, 5A00H, 5A01H                                      | システム中で利用できるページ数が、新たなアロケート要求には不十分である。プログラムは、より少ないページ数を要求することで、この状態から回復できる。                                                                                    |
| 88H | 43H, 51H, 5A00H, 5A01H                                      | 現在利用可能なアロケートされていないページ数では、割り当て要求を受け入れるには不十分である。プログラムは、利用可能なページ数が十分になったとき、もう一度要求するか、より少ないページを要求することによってこの状態から回復できる。                                            |
| 89H | なし                                                          | このエラーコードはメモリマネージャのバージョン4.0か、それ以後のバージョンでは返されない。メモリマネージャの初期バージョンでは、このコードはゼロページがハンドルに割り当てることができないことを示した。この状態はもはや存在しない。                                          |
| 8AH | A4H, 5000H, 5001H, 5500H, 5501H, 5601H, 5700H, 5701H        | メモリにマップする論理ページが、ハンドルに割り当てられている論理ページの範囲外にある。プログラムは、ハンドルの限界内にある論理ページをマップすることによって、この状態から回復できる。                                                                  |
| 8BH | 44H, 4F00H, 4F02H, 5000H, 5001H, 5600H, 5601H, 5500H, 5501H | 1ページ以上の物理ページが、許容可能な物理ページの範囲外にある。物理ページの数は一時的に0から番号づけされている。プログラムは、システムがサポートしている物理ページにマッピングすることによってこの状態から回復できる。                                                 |
| 8CH | 47H                                                         | マップレジスタコンテキスト保存域がいっぱいである。プログラムは再びマップレジスタを保存することを試みることによって、この状態から回復できる。                                                                                       |

|     |                                  |                                                                                                                                                                                              |
|-----|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 8DH | 47H                              | マップレジスタコンテキストスタックには、すでにハンドルに関連したコンテキストが存在する。プログラムは、ハンドルに対するコンテキストがスタックにすでに存在しているとき、マップレジスタコンテキストを保存することを試みた。プログラムは再びコンテキストを保存しなければ、この状態から回復できる（これは、ハンドルのスタック上のマップレジスタコンテキストが正しいことを仮定している）。   |
| 8EH | 48H                              | マップレジスタコンテキストスタックには、指定されたハンドルに関係があるコンテキストは存在しない。プログラムは、ハンドルに対するコンテキストがスタックに存在しないとき、マップレジスタコンテキストを復元しようとした。プログラムは再びコンテキストを復元しようとしなければ、この状態から回復できる（これは、現在のマップレジスタコンテキストが正しいことを仮定している）。         |
| 8FH | 部分ファンクションコードを必要としている、すべてのファンクション | ファンクションに渡される部分ファンクションパラメータが定義されていない。                                                                                                                                                         |
| 90H | 5201H                            | 属性の型が未定義である。                                                                                                                                                                                 |
| 91H | 5200H, 5201H                     | システム構成が、不揮発性をサポートしていない。                                                                                                                                                                      |
| 92H | 5700H                            | 拡張メモリの、ソース領域とコピー先の領域が同じハンドルを持ち、重なりあっている。この場合にもコピーは有効である。コピーが完了すると、少なくともソース領域の一部がコピーによって重ね書きされてしまう。異なるハンドルをもつ拡張メモリのソース領域とコピー先の領域は、決して物理的に重なり合わないよう注意すること。なぜなら、異なるハンドルは、拡張メモリの異なる領域を指定するからである。 |
| 93H | 5700H, 5701H                     | 拡張メモリの指定されたソース領域か、コピー先の領域の大きさが、相手の領域より大きい。指定された大きさの領域をコピー／交換するには、このハンドルにアロケートされたページでは不十分である。プログラムは、コピー先かコピー元のハンドルにページを追加してアロケートするか、または                                                       |



指定した大きさを減らすことによって、この状態から回復する。

しかし、もしアプリケーションプログラムに、それが必要とするメモリと同じ大きさの拡張メモリが割り当てられていれば、それはプログラムエラーであり、回復不可能である。

94H 5700H, 5701H

標準メモリ領域と拡張メモリ領域が、重なりあっている。これは無効である。標準メモリ領域は、拡張メモリ領域と重なり合うことができない。

95H 5700H, 5701H

論理ページ内にあるオフセットが、論理ページの大きさよりも大きい。拡張メモリ領域内にあるコピー元やコピー先のオフセットは、0 から (論理ページ-1)、あるいは 0 から 16383 (3FFFH) の間になければならない。

96H 5700H, 5701H

領域の大きさが 1 メガバイトを超えている。

97H 5701H

拡張メモリのソース領域と交換先の領域が同じハンドルを持ち、重なり合っている。これは無効である。拡張メモリのソース領域と交換先の領域が交換されているとき、同じハンドルを持って重なり合うことができない。異なったハンドルをもつ拡張メモリのソース領域と交換先の領域は、決して物理的に重なり合わないよう注意すること。なぜなら異なったハンドルは拡張メモリの異なった領域を指定するからである。

98H 5700H, 5701H

元のメモリと目的のメモリの型が未定義である／サポートされていない。

9AH 5B01H, 5B06H,  
5B07H

代替用マップレジスタセットがサポートされているが、指定された代替用マップレジスタはサポートされていない。

9BH 5B03H, 5B05H

代替用マップ／DMA レジスタセットはサポートされているが、すべての代替用マップ／DMA レジスタセットが現在アロケートされている。

9CH 5B01H, 5B04H,  
5B06H, 5B07H,  
5B08H

代替用マップ／DMA レジスタセットはサポートされていない。また指定された代替用マップ／DMA レジスタセットはゼロでない。



|     |                                                                                                             |                                                                                                        |
|-----|-------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| 9DH | 5B01H, 5B04H,<br>5B06H, 5B07H,<br>5B08H                                                                     | 代替用マップ/DMAレジスタセットはサポートされているが、指定された交替用マップレジスタセットは、定義されていないか、アロケートされていないか、あるいは現在アロケートされているマップレジスタセットである。 |
| 9EH | 5B06H, 5B07H                                                                                                | 専用のDMAチャンネルがサポートされていない。                                                                                |
| 9FH | 5B06H, 5B07H                                                                                                | 専用のDMAチャンネルはサポートされているが、指定したDMAチャンネルはサポートされていない。                                                        |
| A0H | 5401H                                                                                                       | 指定されたハンドル名に対するハンドル値が見つからない。                                                                            |
| A1H | 5301H, 5401H                                                                                                | この名前をもつハンドルはすでに存在している。指定されたハンドルは名前が割り当てられていない。                                                         |
| A2H | 5700H, 5701H                                                                                                | コピー/交換中に1メガバイトのアドレス空間を超えようとした。コピー/交換する領域の大きさまたはソースの先頭アドレスが1メガバイトを超えている。データはコピー/交換されていない。               |
| A3H | 4E01H, 4E02H,<br>4F00H, 4F01H,                                                                              | ファンクションに渡されたデータ構造の内容が不正に取り扱われているか、意味のないものである。                                                          |
| A4H | 5900H, 5B00H,<br>5B01H, 5B02H,<br>5B03H, 5B04H,<br>5B05H, 5B06H,<br>5B07H, 5B08H,<br>5D00H, 5D01H,<br>5D02H | オペレーティングシステムが、このファンクションのアクセスを拒否した。このファンクションはこの時点で使用できない。                                               |

## 5.6 拡張メモリマネージャの有無のテスト

アプリケーションプログラムが拡張メモリマネージャを使用する前に、MS-DOS は拡張メモリマネージャをロードしたかどうかを調べなければなりません。ここでは、プログラム中で、メモリマネージャの有無を調べるための2つの方法——MS-DOS の“open handle”法と MS-DOS の“get interrupt vector”法——を説明します。また、どのような状況でどちらの方法が適切であるかを説明します。

### 5.6.1 プログラムでどちらの方法を用いるとよいか

ほとんどのアプリケーションプログラムでは、“open handle”法も“get interrupt vector”法も使用することができます。しかし、扱うプログラムが、デバイスドライバになるか、ファイルシステム操作中にMS-DOSに割り込みをかける場合は、“get interrupt vector”法を使用しなければなりません。

デバイスドライバは、MS-DOS 内部から実行されるので、MS-DOS ファイルシステムにアクセスすることができません。ファイルシステムの操作中に MS-DOS に割り込みをかけるプログラムでも、類似した制限があります。割り込み処理の手続きが行われている間は、プログラムは MS-DOS ファイルシステムにアクセスすることができません。なぜならば、他のプログラムが、そのファイルシステムを使用している可能性があるためです。“get interrupt vector”法は MS-DOS のファイルシステムを必要としないので、このようなタイプのプログラムに対しては、この方法を使用しなければなりません。

### 5.6.2 “open handle”法

大部分のアプリケーションプログラムは、MS-DOS の“open handle”法を使用して、メモリマネージャの有無を調べることができます。ここでは、この方法を例とともに説明します。

**警告：**扱うプログラムが、デバイスドライバになるか、ファイルシステムの操作中に MS-DOS に割り込みをかける場合は、この方法を使用してはいけません。このようなプログラムでは、“get interrupt vector”法を使用してください。

#### ■ “open handle”法の使用法

ここでは、MS-DOS の“open handle”を使用して、メモリマネージャの有無を調べる方法を説明します。この方法は、次の手順で行います。

① “read only”のアクセスモード（レジスタ AL=0）で、“open handle”コマンド（MS-DOS のファンクション 3DH）を実行します。このファンクションは、目的とするファイルやデバイスのパス名を含む ASCII 文字列を指示するためのプログラムが必要です（レジスタセット DS:DX がポインタを含む）。この場合には、そのファイルは実際のメモリマネージャの名前です。



次のような形式の ASCII 文字列を使用します。

ASCII\_device\_name DB "EMMXXXX0",0

大文字 EMMXXXX0 に対する ASCII コードは 00H (NULL) で終わります。

② MS-DOS がエラーステータスコードを返さないときには、③、④を省略して⑤へ進みます。エラーステータスコードが "Too many open files" であった場合は、③へ進みます。エラーステータスコードが "File / Path not found" であった場合は、③を省略して④へ進みます。

③ MS-DOS が、ハンドルが充分でないことを示す "Too many open handle" ステータスコードを返した場合には、他のファイルをオープンする前に、ユーザーは扱うプログラムで "close file" コマンドを実行するべきです。これによって、少なくとも 1 つのファイルハンドルを、このエラーを引き起こさずにファンクションの実行で利用できることが保証されます。

プログラムが "open file" コマンドを実行した後では、⑥で説明するテストを実行し、"file handle" をクローズするべきです (MS-DOS ファンクション 3EH)。"manager" ファンクションは、MS-DOS を通じて利用することができないので、このステータステストの実行後も、そのマネージャをオープンしたままにしておくべきではありません。⑥へ進みます。

④ MS-DOS が "File / Path not found" を返す場合には、そのメモリマネージャはインストールされていません。アプリケーションがメモリマネージャを必要としているのであれば、ユーザーは、メモリマネージャと適切な CONFIG.SYS ファイルを収めているディスクでシステムをブートしなおさなければなりません。

⑤ MS-DOS がエラーステータスコードを返さない場合は、"EMMXXXX0" という名前のデバイスがシステムに常駐しているか、同じ名前のファイルが現在のディスクドライブ装置内に存在していると考えられます。⑥へ進みます。

⑥ "get device information" コマンド (レジスタ AL=00H) を持つ "I / O Control for Device" コマンド (MS-DOS ファンクション 44H) を実行し、EMMXXXX0 がデバイスであるかファイルであるかを調べます。

ユーザーは、"EMM" デバイスにアクセスするために、①で取得したファイルハンドル (レジスタ BX) を使用しなければなりません。

このファンクションは、"device information" を 1 ワード (レジスタ DX) に返します。⑦へ進みます。

⑦ MS-DOS が、なんらかのエラーステータスコードを返す場合には、メモリマネージャデバイスドライバがインストールされていないことが考えられます。もし、アプリケーションがメモリマネージャを必要とするのであれば、ユーザーは、メモリマネージャと適切な CONFIG.SYS



ファイル収めているディスクで、システムをブートしなおさなければなりません。

⑧もし MS-DOS がエラーステータスを返さなければ、ファンクションが返した "device information" ワード (レジスタ DX) の第 7 ビット (0 から数えて) を調べます。⑨へ進みます。

⑨もし "device information" ワードの第 7 ビットが 0 (ゼロ) であったならば、"EMMXXX0" はファイルであり、メモリマネージャデバイスドライバは存在しません。もし、アプリケーションがメモリマネージャを必要とするのであれば、ユーザーは、メモリマネージャと適切な CONFIG.SYS ファイル収めているディスクで、システムをブートしなおさなければなりません。

もし第 7 ビットが 1 であったならば、EMMXXX0 はデバイスです。⑩へ進みます。

⑩ "get output status" コマンド (レジスタ AL=07H) を持つ "I / O Control for Device" コマンド (MS-DOS ファンクション 44H) を実行します。

ユーザーは、"EMM" デバイスにアクセスするために、①で取得したファイルハンドル (レジスタ BX) を使用しなければなりません。⑪へ進みます。

⑪もし、拡張メモリデバイスドライバが "ready" ならば、メモリマネージャはレジスタ AL にステータス値 "0FFH" を渡します。もし、デバイスドライバが "not ready" ならば、ステータス値は "00H" です。

もしメモリマネージャデバイスドライバが "not ready" で、アプリケーションがそれを必要としているならば、ユーザーはメモリマネージャと適切な CONFIG.SYS ファイル収めたディスクでシステムをブートしなおさなければならぬでしょう。

もし、メモリマネージャデバイスドライバが "ready" であれば、⑫へ進みます。

⑫拡張メモリデバイスドライバをクローズするために "Close File Handle" コマンド (MS-DOS ファンクション 3EH) を実行します。ユーザーは、"EMM" デバイス (レジスタ BX) をクローズするために、①で取得したファイルハンドルを使用しなければなりません。

## ■ “open handle” 法の使用例

次の手続きは、前述の “open handle” 法の使用例です。

次の手続きは、システム中の EMM の存在を調べます。

もし、EMM が存在すれば、CARRY FLAG SET を返します。

もし、EMM が存在しなければ、CARRY FLAG CLEAR を返します。

```
first_test_for_EMM PROC NEAR
```

```
PUSH DS
```

```
PUSH CS
```

```
POP DS
```

```
MOV AX,3D00H ; “read only” モードで “device open” を実行す
; る
```

```
LEA DX,ASCII_device_name
```

```
INT 21H
```

```
JC first_test_for_EMM_error_exit ; “device open” 中のエラーを調べる
```

```
MOV BX,AX ; DOS によって返された “file handle” を得る
```

```
MOV AX,4400H ; “IOCTL” を実行する
```

```
INT 21H ; “get device info.”
```

```
JC first_test_for_EMM_error_exit ; “get device info.” 中のエラーを調べる
```

```
TEST DX,0080H ; ASCII_device_name がデバイスかファイルか
; を調べる
```

```
JZ first_test_for_EMM_error_exit
```

```
MOV AX,4407H ; “IOCTL” を実行する
```

```
INT 21H
```

```
JC first_test_for_EMM_error_exit ; “IOCTL” 中のエラーを調べる
```

```
PUSH AX ; “IOCTL” ステータスを保存する
```

```
MOV AH,3EH ; “CLOSE FILE HANDLE” を実行する
```

```
INT 21H
```

```
POP AX ; “IOCTL” ステータスを復元する
```

```
CMP AL,0FFH ; ドライバによって返された “DEVICE READY”
; ステータスを調べる
```

```
JNZ first_test_for_EMM_error_exit
```

```
first_test_for_EMM_exit:
```

```
POP DS ; EMM がシステムに存在する
```

```
STC
```



```
RET
```

```
first_test_for_EMM_error_exit:
```

```
POP DS ; EMM はシステムに存在しない
```

```
CLC
```

```
RET
```

```
ASCII_deice_name DB "EMMXXXXX0",0
```

```
first_test_for_EMM ENDP
```

### 5.6.3 “get interrupt vector” 法

どのようなタイプのプログラムでも、MS-DOS の “get interrupt vector” 法を用いてメモリマネージャの有無を調べることができます。ここでは、その方法と、使用例を説明します。

**警告：**扱うプログラムが、デバイスドライバであるか、ファイルシステムの操作中に MS-DOS に割り込みをかけるならば (“open handle” 法を使用しないで)、必ずこの方法を使用してください。

#### ■ “get interrupt vector” 法の使用方法

ここでは、メモリマネージャの有無を調べる MS-DOS の “get interrupt vector” 法を説明します。この方法は、次の手順で行います。

① “get vector” コマンド (MS-DOS ファンクション 35H) を実行して、割り込みベクトルのエントリ番号 67H の内容を取得します (0000:019C~0000:019F のアドレス)。

メモリマネージャは、どのマネージャファンクションを実行するにも、この割り込みベクトルを使用します。この割り込みサービスルーチンのアドレスのオフセット部は、アドレス 0000:019C にあるワード中に、セグメント部はアドレス 0000:019E にあるワード中に保存されます。

②割り込みベクトルアドレス 67H のセグメント部と固定オフセット 000AH で指定されるアドレスで始まる ASCII 文字列の内容と、“device name field” とを比較します。もし DOS が、ブート時にメモリマネージャをロードしていれば、この “device name field” にはデバイス名が入ります。

メモリマネージャはキャラクタデバイスドライバとしてインプリメントされているので、プログラムは 0000H から始まります。デバイスドライバは、その位置に位置づけられた “device header” を持っています。“device header” には、8 バイトの “device name field” があります。キャラクタモードデバイスドライバにおいては、この “device name field” はいつも device header 内のオフセット 000AH に位置づけられます。“device name field” は、MS-DOS がデバイスを参照するときに、そのデバイス名を含んでいます。



もし、この方法での“文字列比較”の結果が正しければ、メモリマネージャデバイスが存在します。

#### ■ “get interrupt vector” 法の使用例

次の手続きは、前述の“get interrupt vector”法の使用例です。

次の手続きは、システム中の EMM の存在を調べます。

もし、EMM が存在すれば、CARRY FLAG SET を返します。

もし、EMM が存在しなければ、CARRY FLAG CLEAR を返します。

```
second_test_for_EMM PROC NEAR
PUSH DS
PUSH CS
POP DS
MOV AX,3567H ; “get interrupt vector” を実行する
INT 21H
MOV DI,000AH ;DOS で返された ES の SEGMENT を使用する。
 ;DI での “device name field” オフセットを配置する
LEA SI,ASCII_device_name ;SI で装置名文字列のオフセットを配置する。
 ;SEGMENT はすでに DS 内にある
MOV CX,8 ;名前文字列を比較する
CLD
REPE CMPSB
JNE second_test_for_EMM_error_exit
second_test_for_EMM_exit:
POP DS ;EMM がシステムに存在する
STC
RET
second_exit_for_EMM_exit:
POP DS ;EMM はシステムに存在しない
CLC
RET
ASCII_device_name DB “EMMXXXX0”
second_test_for_EMM ENDP
```

## 5.7 拡張メモリマネージャのインプリメンテーションへのガイドライン

ファンクションの仕様に加えて、拡張メモリマネージャはある資源を用意しています。つぎのような必要とされる資源が、EMS のこのバージョンに従う拡張メモリマネージャに存在するようにガイドラインが用意されています。

### サポートされる拡張メモリの量

拡張メモリの量は、最大 32 メガバイトまでです。

### サポートされるハンドル数

拡張メモリのハンドル数の、最大数は 255 で、最小数は 64 です。

### ハンドルの番号づけ

ハンドルは 1 ワードの大きさですが、オペレーティングシステムのハンドル数も含めて、最大 255 個あります。この EMS では、次のようなハンドルワードを定義しています。ワードはメモリマネージャによって、下位バイトは実ハンドル値、上位バイトは 00 にセットされています。この EMS の前のバージョン (3.2 以前) では、ハンドルの値を指定していませんでした。

### 新しいハンドルの型：ハンドルとローハンドル

ロー (raw) ハンドルと標準的なハンドルの差異は、ほとんどありません。もし、Allocate raw pages ファンクション (ファンクション 27) を使用すれば、DX で返されるものがローハンドルです。ローハンドルは必ずしも 16K バイトとは限りません。"ロー (raw)" ページの大きさに関係しないかわりに、それは拡張メモリハードウェアに依存するローページサイズとなります。

アプリケーションプログラムは、ファンクション 26 によって、ローページの大きさを調べることができます。また、ファンクション 27 が返すローハンドルを使用することによって、特別な拡張メモリボードだけが許容する、より優れた方法でアクセスすることができます。

一方、allocate page ファンクション (ファンクション 4) を使用するアプリケーションは、必ず 16K バイトを扱うようなハンドルを割り当てられます。より小さなローページを持つ拡張メモリボードに関しては、EMM デバイスドライバが、ひとつの合成した 16K バイトのページを作るために必要とするローページ数を割り当て、保持します。拡張メモリボードのローページの大きさと 16K バイトの EMS 標準ページの大きさとの差異は、アプリケーションファンクション 4 で得られるハンドルを使用しているとき、そのアプリケーションには明らかです。

メモリマネージャは、ハンドルにアロケートされたページと、ローハンドルにアロケートされたページを区別しなければなりません。ドライバへの呼び出しの意味は、メモリマネージャに、ハンドルが渡されるかローハンドルが渡されるかということによって変わります。たとえ



ば、もしハンドルがファンクション 18 (リアロケート) に渡されれば、メモリマネージャはハンドルに割り当てられた 16K バイトのページ数を増加/減少させるでしょう。もし、ファンクション 18 がローハンドルを渡されれば、メモリマネージャはローハンドルに割り当てられたローページ (16K バイトではない) を増加/減少させるでしょう。EMS 標準ボードにおいては、ページとローページの違いはありません。

#### システムローハンドル (ローハンドル=0000H)

640K バイトよりも下のアドレス空間にあるメモリをマップし直すことが可能な拡張メモリボードでは、640K 以下の部分に位置するメモリのページを管理することが、ひとつの問題になります。メモリマネージャはこの問題を解決するために、MS-DOS がマネージャをロードするときに値 0000H を持つローハンドルを作ります。このローハンドルは、システムハンドルと呼ばれます。

さらに極端な場合、メモリマネージャは 640K バイト以下にマップされる全ページをシステムハンドルに割り当てるでしょう。これらのページは、論理的順序でマップされるべきです。たとえば、システムボードが 256K バイトの RAM をサポートし、それよりも 384K バイト上位がマップ可能であれば、そのシステムハンドルはその論理ページ 0 を 256K にある第 1 物理ページに、論理ページ 1 を次の物理ページにというようにマップするべきです。

システムハンドルは、ローページを扱うべきです。アプリケーションプログラムが使用できるようにこれらのページを何ページか解放するために、オペレーティングシステムが "Reallocate" ファンクションも使用してシステムハンドルに割り当てられたページ数を減少させることができます。"Deallocate" ファンクションを実行することはシステムハンドルを大きさ 0 にしますが、ローハンドル自体のアロケートを解除するわけではありません。"Deallocate" ファンクションは、他のローハンドルとシステムハンドルを区別して扱います。もしオペレーティングシステムが以前に "exited" されているならば (たとえば MS-WINDOWS が "exited" しているように)、元のシステムハンドルを 640K バイトを満たすように増加させ、DOS に返す前に元の論理ページを物理メモリにマッピングします。

このハンドルに対して、機能上 2 つの特別な場合があります。

第 1 の特別な場合は、ファンクション 4 (Allocate Pages ファンクション) を扱います。このファンクションは、ハンドルの値として 0 (ゼロ) を返すことはありません。アプリケーションはいつもファンクション 4 を呼び出して、ページをアロケートし、ページを取り扱うハンドルを取得しなければなりません。ファンクション 4 は、決してハンドル値 0 を返しませんので、アプリケーションはこの特別なハンドルへアクセスすることはできません。



第2の特別な場合は、ファンクション 6 (Deallocate Pages ファンクション) を扱います。もし、オペレーティングシステムがファンクション 6 を使用してシステムハンドルにアロケートされているページを解除すると、そのページはマネージャに返され、再利用できますが、ハンドルを再び割り当てることはできません。マネージャは、このハンドルに対する“deallocate pages” ファンクションの要求を“reallocate pages” ファンクションの要求と同じものとして扱います。ここでこのハンドルに割り当てられるページ数は 0 です。

## 5.8 ファンクション 28 におけるオペレーティングシステムの利用

拡張メモリボードには、論理ページから物理ページへのマップを“記憶する”1組のレジスタがあります。いくつかのボードには、数組の特別な（代替用）マップレジスタがあります。拡張メモリボードは、代替用マップレジスタの組を限られた数しか提供することはできないので、この仕様ではファンクション 28 (Alternate Map Register ファンクション) を使用して、それらをシミュレートする方法を提供しています。

### ■使用例

ここでの例は、ハードウェアが代替用マップレジスタセットをサポートしていることを想定しています。第1ウィンドウを持ってきて、それから“Reversi”が始まります。そのとき、制御は MS-DOS 監視プログラムに返り、切り換わります。この手続きのために拡張メモリマネージャへの呼び出しがあります。

#### 例 1

|                            |                      |
|----------------------------|----------------------|
| Allocate alt reg set       | ; MS-DOS をスタートする     |
| (for the MS-DOS executive) | ; 監視プログラム            |
| Set all reg set            |                      |
| (for MS-DOS Executive)     |                      |
| Allocate alt reg set       | ; Reversi をスタートする    |
| (for Reversi)              |                      |
| Set alt reg set            |                      |
| (for Reversi)              |                      |
| Map pages                  |                      |
| (for Reversi)              |                      |
| Set alt reg set            | ; MS-DOS へ返って切り換えられる |
| (for MS-DOS Executive)     | ; 監視プログラム            |

例 2

|                              |                    |
|------------------------------|--------------------|
| Allocate alt reg set         | ; MS-DOS をスタートする   |
| (for the MS = DOS executive) | ; 監視するプログラム        |
| Set alt reg set              |                    |
| (for MS-DOS Executive)       |                    |
| Get alt reg set              |                    |
| (for MS = DOS Executive)     |                    |
| Allocate alt reg set         | ; Reversi をスタートする  |
| (for Reversi)                |                    |
| Set alt reg set              |                    |
| (for Reversi)                |                    |
| Map pages                    |                    |
| (for Reversi)                |                    |
| Get alt reg set              |                    |
| (for Reversi)                |                    |
| Set alt reg set              | ; MS-DOS へ返って切り換える |
| (for MS-DOS Executive)       | ; 監視プログラム          |

考慮されなければならない重要な点は、Set は必ず Get に先行しなければならないということです。Set のあとの Get という形式は、割り込みハンドラが使用する Get のあとで Set という形式（古いマップコンテキストを得て、新しいものをセットする）の逆になります。他の重要な点は、代替用マップレジスタのセットは、割り当てられるとき、現在のマッピングを持たなければならない、そうでないときはその Set は混乱をきたすことになるということです。

これがソフトウェアでシミュレートされれば、いったい何が起きるのでしょうか。同じ Set と Get のモデルが当てはまります。主な相違は、コンテキストが保存される場所です。

アロケート呼び出しが動的で、割り当てられている組数に制限がないので、オペレーティングシステムは要求される空間を提供しなければなりません。その要求は失敗するので、デバイスドライバは、動的に空間をアロケートすることができません。もしアロケートレジスタセットの呼び出しが、代替用マップレジスタセットをサポートしていないというステータスを返すならば、オペレーティングシステムはそのコンテキストに対して空間をアロケートしなければ



なりませんし、また、ファンクション 15 を用いてコンテキストを初期化しなければなりません。その時点でオペレーティングシステムは、Set をマッピングコンテキスト空間に対するポインタを渡すことで、実行することができます。Get の呼び出しに関しては、EMM デバイスドライバにその同じコンテキストの空間に対するポインタを返します。

### 例 3

|                            |                    |
|----------------------------|--------------------|
| Allocate alt reg set       | ; MS-DOS をスタートする   |
| (for the MS-DOS executive) | ; 監視プログラム          |
| Get Page Map               |                    |
| (for the MS-DOS executive) |                    |
| Set alt reg set            |                    |
| (for MS-DOS Executive)     |                    |
| Allocate alt reg set       | ; Reversi をスタートする  |
| (for reversi)              |                    |
| Set alt reg set            |                    |
| (for Reversi)              |                    |
| Map pages                  |                    |
| (for Reversi)              |                    |
| Get Page Map               |                    |
| (for Reversi)              |                    |
| Set alt reg set            | ; MS-DOS へ返って切り換える |
| (for MS-DOS Executive)     | ; 監視プログラム          |

## 5.9 用 語

この解説で数多く使用された用語をいくつか説明します。

### アプリケーションプログラム (Application Program)

アプリケーションプログラムとは、プログラマーが作成し、ユーザーが使用するプログラム。アプリケーションソフトウェアの種類には、ワードプロセッサやデータベースマネージャ、スプレッドシートマネージャ、プロジェクトマネージャなどがある。

### EMM (イーエムエム)

拡張メモリマネージャを参照。

### 拡張メモリ (Extended Memory)

保護仮想アドレスモードで操作されているとき、80286, 386/386SX プロセッサ上で利用可能



な100000HからFFFFFFHまでの15メガバイトの範囲アドレスを持つメモリ。PC-9800シリーズの“拡張メモリ”はこのExtended Memoryのことを指す。

#### 拡張メモリマネージャ (Expanded Memory Manager, EMM)

MS-DOS のアプリケーションプログラムと拡張メモリのインターフェイスを制御するデバイスドライバ。

#### ローページ (Raw Page)

拡張メモリボードが与えることができる、マップ可能な最小単位のメモリ。

#### ステータスコード (Status Code)

EMM ファンクションが値として返すコードは、そのファンクションを実行した結果に関するなんらかの情報を示すものである。ステータスコードには、そのファンクションが正確に実行されたかどうかを示すものや、拡張メモリハードウェア、ソフトウェアについての情報を示すものがある。

#### 常駐アプリケーションプログラム (Resident Application Program)

常駐アプリケーションプログラムは MS-DOS によってロードされ、実行され、MS-DOS に制御を戻した後でも、依然としてシステムに常駐するプログラムである。このタイプのプログラムはメモリを占有し、通常、オペレーティングシステムやアプリケーションプログラムやハードウェアによって呼び出される。常駐アプリケーションプログラムの例としては、RAM ディスクドライバ、プリントスプーラ、“ポップアップ”デスクトッププログラムなどがあげられる。

#### 通常メモリ (Conventional Memory)

アドレス 00000H から 9FFFFH にある、0～640K バイトのメモリ。

PC-9800 シリーズのハイレゾリューションモードでは、00000H～BFFFFH の 768K バイトのメモリ。

#### アロケート (Allocate)

拡張メモリのページを、指定した大きさだけ確保すること。

#### デアロケート (Deallocate)

以前にアロケートした拡張メモリを、メモリマネージャに返すこと。

#### ハンドル (Handle)

EMM が、アプリケーションプログラムで用いられる 1 ブロックのメモリを識別するために割り当てて使用する値。割り当てられたすべての論理ページは、個々のハンドルと関連がある。

**非常駐アプリケーションプログラム (Transient Application Program)**

非常駐アプリケーションプログラムは、MS-DOS によってロードされ、実行され、制御を MS-DOS に返した後に、そのシステムには常駐しないプログラムである。非常駐アプリケーションプログラムが MS-DOS に制御を移した後は、そのプログラムが使用していたメモリを他のプログラムが利用できる。

**物理ページ (Physical Page)**

ある 1 つの 16K バイトの大きさを持つページによって占められるメモリアドレスの範囲。

**ページフレーム (Page Frame)**

隣接した 16K バイトの物理ページの集まりで、アプリケーションプログラムはこれを介して拡張メモリをアクセスする。

**ページフレーム基底アドレス (Page Frame Base Address)**

ページフレーム基底アドレスは、ページフレームの最初のバイトの位置 (セグメント形式) である。

**マッピング (Mapping)**

物理ページ上に現われる、メモリの論理ページを作る処理。

**マッピングコンテキスト (Mapping Context)**

ある特定の時点でのマップレジスタの内容。このコンテキストは、マップの状態を表わしている。

**マップ解除 (Unmap)**

論理ページに対して、読み込み／書き出しのアクセスを不可能にすること。

**マップ可能なセグメント (Mappable Segmen)**

マップされた論理ページを持つことが可能な、16K バイトのメモリ領域。

**マップレジスタ (Map Registers)**

EMM ハードウェアの現在のマッピングコンテキストを含んでいる 1 組のレジスタ。

**論理ページ (Logical Page)**

EMM は、拡張メモリを論理ページと呼ばれる (典型的には 16K バイト) 単位にアロケートする。





## 第6章

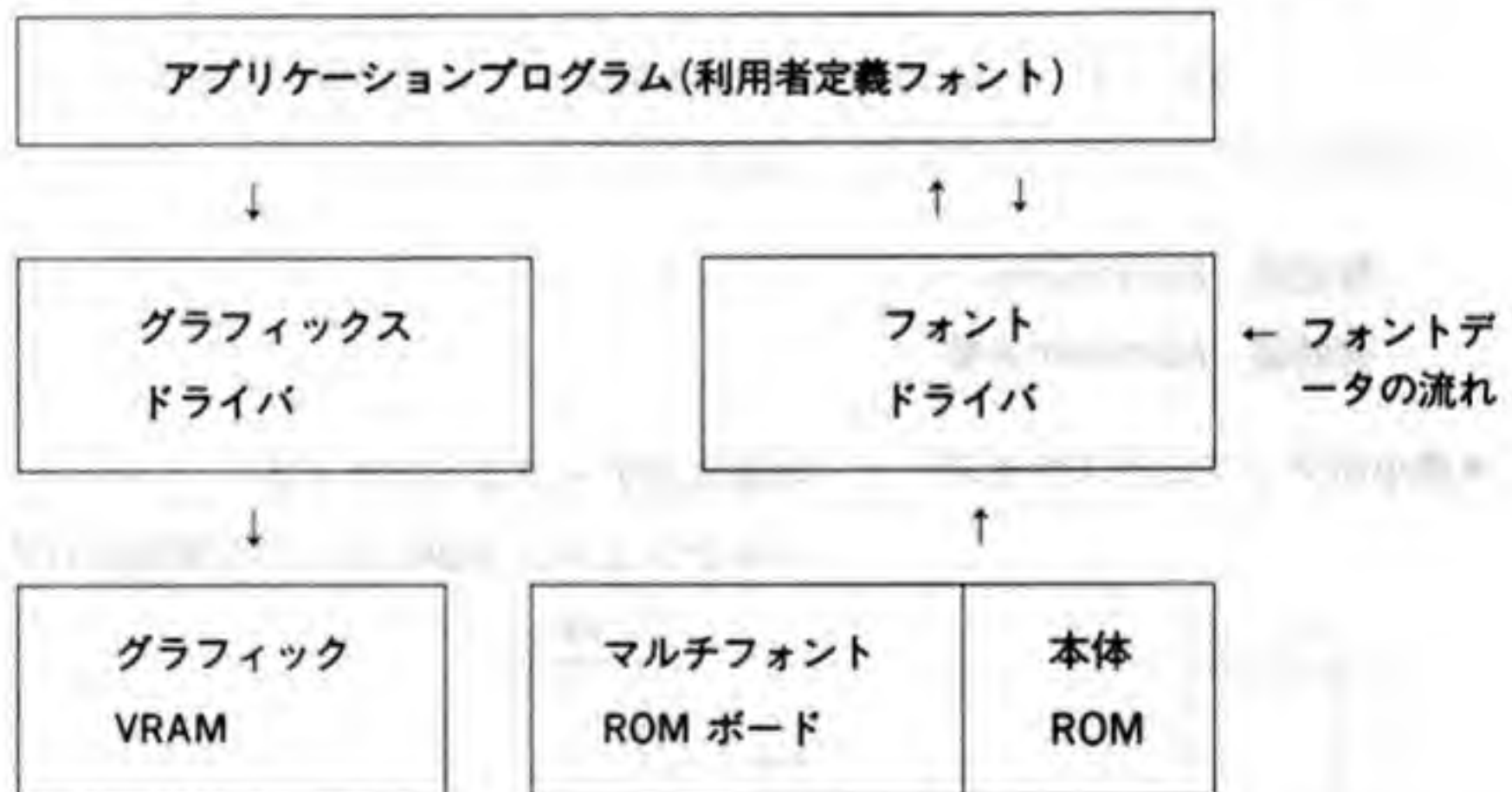
## 6.1 イントロダクション

フォントドライバは、マルチフォント ROM ボードや本体 ROM などを利用して2バイト JIS コードの文字フォントを編集したり、利用者定義フォントを拡大/縮小し、ユーザー領域に出力するデバイスドライバです。

得られたフォントは、グラフィックスドライバを利用することにより、グラフィックVRAMに出力することができます。

この章ではフォントドライバの各機能について説明します。

## プログラム関係図



注意：PC-9801, PC-9801E, PC-9801F, PC-9801M ではマルチフォント ROM ボードが使用できないので、本体 ROM のみのサポートとなります。また、PC-98LT では動作しません。

## 6.2 フォントドライバ

ここでは、フォントドライバの組み込みと、利用できるファンクション(機能)の一覧を紹介します。

### 6.2.1 フォントドライバの組み込み

フォントドライバを利用するには、CUSTOM コマンドやエディタにより、CONFIG.SYS ファイルに次のような一行を加えてシステムを再起動します。

**DEVICE=[パス]FONT.SYS[/M(nnn,mmm)][/E]**

/M スイッチは取得する文字フォントの最大ボディフェイスサイズをX方向のドット数(nnn)とY方向のドット数(mmm)で指定します。有効値は下記の通りですが有効値範囲外またはデフォルトの場合は、X 方向ドット数 40 ドット、Y 方向ドット数 40 ドットとなります。

有効値： $8 \leq nnn \leq 400$

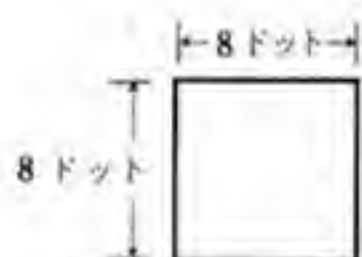
有効値： $8 \leq mmm \leq 400$

ただし、マルチフォント ROM ボードが実装されていない場合の最大ボディフェイスサイズの有効値は以下のようになります。ご注意ください。

有効値： $8 \leq nnn \leq 40$

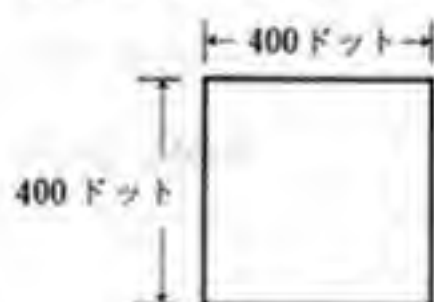
有効値： $8 \leq mmm \leq 40$

#### ●最小ボディフェイスサイズ

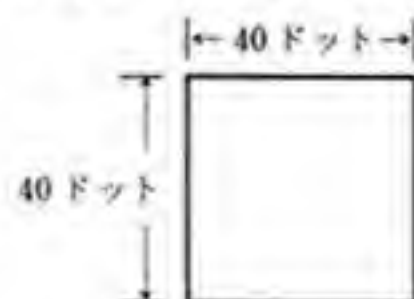


#### ●最大ボディフェイスサイズ

マルチフォント ROM ボードが実装されている場合



マルチフォント ROM ボードが実装されていない場合



/E スイッチは EMS ドライバが組み込まれているとき、フォントドライバの一部を拡張メモリに転送し、MS-DOS のメモリを節約します。ただし、/E スイッチを指定すると、フォントイメージ取得アドレス、フォントドライバ作業域アドレスに拡張メモリは指定できません。

EMS ドライバはバージョン 3.x 以上を使用してください。

### 6.2.2 ファンクション一覧

- 環境設定ファンクション

フォント情報の設定

- フォント情報取得ファンクション

バージョンの取得

フォント情報の取得

- 文字フォント取得ファンクション

フォントの取得

## 6.3 フォントドライバの呼び出し方法と使用例

ここでは、ユーザープログラムでフォントドライバのファンクションを利用する際の呼び出し手順と、マクロアセンブラでの使用例を紹介します。

### 6.3.1 ファンクションの呼び出し手順

フォントドライバの各ファンクションは次の手順で呼び出します。

(1) フォントドライバのエントリテーブルの先頭アドレスの取得

AX レジスタ、DS:BX レジスタを次のようにセットし、INT CCH を行うとエントリテーブルの先頭アドレスが指定したアドレスに設定されます。

AX=0

DS:BX=先頭アドレスを格納する領域（ダブルワード）のアドレス

以下にエントリテーブルの詳細を示します。各テーブルにはそのファンクションのエントリアドレスが格納されています。

| 相対アドレス | エントリアドレス(DWORD) |
|--------|-----------------|
| 0000   | バージョンの取得        |
| 0004   | フォント情報の設定       |
| 0008   | フォント情報の取得       |
| 000C   | フォントの取得         |

(2) ファンクションの呼び出し

パラメータブロック（32 バイト）に必要なパラメータを設定し、この先頭アドレスをスタックに格納後、エントリテーブル内の対応するアドレスに far call します。詳細はファンクションプログラミングインターフェイスを参照してください。



データ領域のアドレス (DWORD) →

|                       |
|-----------------------|
| パラメータブロック<br>32 Bytes |
|-----------------------|

### 6.3.2 マクロアセンブラでの使用例

次にマクロアセンブラ (MASM) によるプログラムの例を紹介します。

```

;
; エントリテーブル内相対アドレスの定義
;
FONTVER EQU 0 ; バージョンの取得
FONTSET EQU 01*4 ; フォント情報の設定
FONTGET EQU 02*4 ; フォント情報の取得
MOJIGET EQU 03*4 ; フォントの取得
;
; データ領域の定義
;
DATA SEGMENT
FONT_ADDR DD 0 ; エントリテーブル先頭アドレス格納域
FONT_PARA_ADDR LABEL DWORD ; パラメータブロックアドレス格納域
FONT_PARA_OFF DW 0 ; オフセットアドレス
FONT_PARA_SEG DW 0 ; セグメントアドレス
FONT_PARA DB 32 DUP(0) ; パラメータブロック領域
FONT_PARA_SIZ EQU 32 ; パラメータブロックサイズ
;
;
FONT_DATA DB 2052 DUP(0) ; フォントデータ格納域
DATA ENDS
;
; スタック領域の定義
;
STACK SEGMENT
 DW 256 DUP(0)
STACK ENDS
;
CODE SEGMENT
 ASSUME CS : CODE, DS : DATA

```

START:

```

MOV AX, DATA
MOV DS, AX
MOV WORD PTR DS: FONT_PARA_SEG, AX
MOV AX, OFFSET FONT_PARA
MOV WORD PTR DS: FONT_PARA_OFF, AX
; エントリテーブル先頭アドレスの取得
MOV AX, 0000H
MOV BX, OFFSET DS: FONT_ADDR
INT OCCH ; エントリテーブル先頭アドレスの取得
; バージョンの取得
LES DI, DS: FONT_PARA_ADDR
PUSH ES
PUSH DI
LES DI, DS: FONT_ADDR
CALL DWORD PTR ES: FONTVER [DI]
; フォント情報の設定
LES DI, DS: FONT_PARA_ADDR
PUSH DI
MOV CX, FONT_PARA_SIZ/2
SUB AX, AX
REP STOSW ; パラメータブロックのクリア
POP DI
MOV WORD PTR ES: 0 [DI], 0 ; フォント種別
MOV BYTE PTR ES: 2 [DI], 0 ; 横書き／縦書きフラグ
MOV WORD PTR ES: 4 [DI], 18 ; X方向ボディフェイスサイズ
MOV WORD PTR ES: 6 [DI], 18 ; Y方向ボディフェイスサイズ
MOV WORD PTR ES: 8 [DI], 2 ; キャラクタフェイス開始X座標
MOV WORD PTR ES: 10 [DI], 2 ; キャラクタフェイス開始Y座標
MOV WORD PTR ES: 12 [DI], 16 ; X方向キャラクタフェイスサイズ
MOV WORD PTR ES: 14 [DI], 16 ; Y方向キャラクタフェイスサイズ
PUSH ES
PUSH DI
LES DI, DS: FONT_ADDR
CALL DWORD PTR ES: FONTSET [DI]

```

;        フォント情報の取得

```

 LES DI, DS : FONT_PARA_ADDR
 PUSH ES
 PUSH DI
 LES DI, DS : FONT_ADDR
 CALL DWORD PTR ES : FONTGET [DI]
 :
 :

```

;        フォントの取得

```

 LES DI, DS : FONT_PARA_ADDR
 PUSH DI
 MOV CX, FONT_PARA_SIZ/2
 SUB AX, AX
 REP STOSW ; パラメータブロックのクリア
 POP DI
 MOV WORD PTR ES:0 [DI], 3441H ; 2バイト日本語 JIS コード
 MOV WORD PTR ES:2 [DI], 58 ; フォントデータ格納域サイズ
 MOV AX, OFFSET DS : FONT_DATA
 MOV WORD PTR ES:4 [DI], AX
 MOV WORD PTR ES:6 [DI], DS ; フォントデータ格納域アドレス
 PUSH ES
 PUSH DI
 LES DI, DS : FONT_ADDR
 CALL DWORD PTR ES : MOJIGET [DI]
 :
 :
CODE ENDS
 END START

```

### 6.3.3 文字フォントの利用例

ここでは、フォントドライバにより取得した文字フォントをグラフィック VRAM に描画する方法を紹介します。

フォントドライバの文字フォントのデータ形式は、グラフィックスドライバの“グラフィックイメージの設定”で利用するモノクロモードのグラフィックイメージデータと同一になっています。したがって、取得した文字フォントのアドレスを格納域アドレス（グラフィックスドライバのパラメータ）に指定して“グラフィックイメージの設定”を実行することにより、文字フォントをグラフィック VRAM に描画します。

## 6.4 ファンクションプログラミングインターフェイス

次にフォントドライバの各ファンクションごとの詳細を説明します。



## バージョンの取得

## ファンクション NO. 0

**機 能** フォントドライバのバージョンを取得します。

**コール** スタック = パラメータブロックの先頭アドレス

**リターン** AX=フォントドライバのバージョン  
           AL: バージョン番号の整数部  
           AH: バージョン番号の小数部

**解 説**       たとえば、バージョン 2.0 では AX に 0002 H が入ります。  
           本ファンクションで、バージョンに 1.0 (AX=0001 H) が返された場合、フォント情報の設定 (ファンクション NO.1) で動作指定フラグに 0 以外を設定することはできません。

# **フォント情報の設定**

# **ファンクション NO. 1**

**機 能**      フォントの種別、フォントサイズなどの情報を指定します。

**コール**      スタック = パラメータブロックの先頭アドレス

## **●パラメータブロック**

| オフセット | サイズ    | 内 容                    |
|-------|--------|------------------------|
| 0H    | WORD   | フォント種別                 |
| 2H    | BYTE   | 横書き／縦書きフラグ             |
| 3H    | BYTE   | 動作指定フラグ                |
| 4H    | WORD   | X方向ボディフェイスサイズ          |
| 6H    | WORD   | Y方向ボディフェイスサイズ          |
| 8H    | WORD   | キャラクタフェイス開始X座標         |
| AH    | WORD   | キャラクタフェイス開始Y座標         |
| CH    | WORD   | X方向キャラクタフェイスサイズ        |
| EH    | WORD   | Y方向キャラクタフェイスサイズ        |
| 10H   | 16BYTE | 未使用（常に 00 H を設定してください） |

## **●フォント種別**

- 0：本体フォント ROM のフォントを利用する。
- 1：マルチフォント ROM ボードのゴシックフォントを利用する。
- 2：マルチフォント ROM ボードの明朝フォントを利用する。
- 3：マルチフォント ROM ボードの 16×16 ドットフォントを利用する。

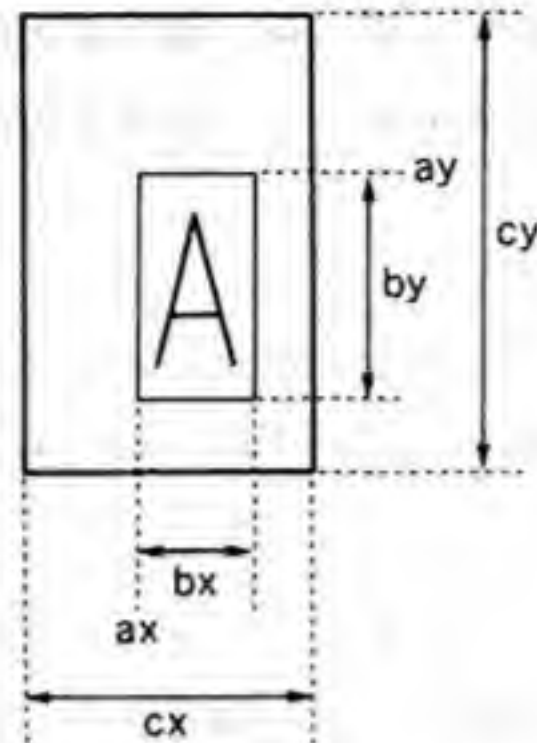
(注) フォント種別に 3 が指定できるのは、マルチフォント ROM ボードが PC-9801-38 L の場合のみです。

## **●横書き／縦書きフラグ**

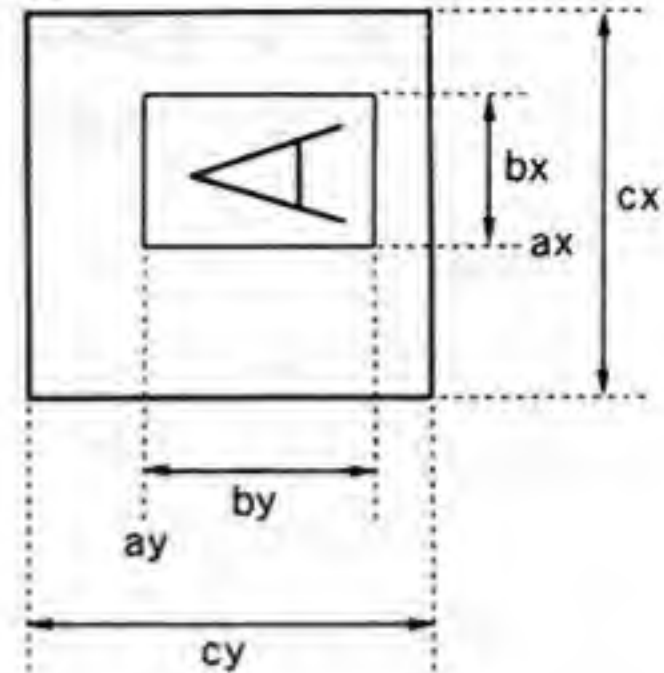
- 0：横書きフォント
- 1：縦書きフォント

縦書きフォントとは、横書きフォント(テキストフォントと同じ向きのフォント)を反時計まわりに 90°回転した書体のフォントです。縦書きフォントの場合は、X 方向と Y 方向の各サイズが入れ換わることに注意してください。

例) 横書きの“A”



縦書きの“A”



ax : キャラクタフェイス開始X座標

bx : X方向キャラクタフェイスサイズ

ay : キャラクタフェイス開始Y座標

by : Y方向キャラクタフェイスサイズ

cx : X方向ボディフェイスサイズ

cy : Y方向ボディフェイスサイズ

## ●動作指定フラグ

フォント取得時の動作を指定する。

0 : 8×8～128×128 ドットフォントの取得(Ver1.0 互換機能)

1 : 8×8～400×400 ドットフォントの取得

2 : 利用者定義フォントを参照して拡大/縮小

(注)動作指定フラグに2を指定した場合、「フォント種別」の設定は無視されます。

## ●X方向ボディフェイスサイズ

取得する文字フォントのX方向ドット数を指定します。

## ●Y方向ボディフェイスサイズ

取得する文字フォントのY方向ドット数を指定します。

## 有効値

フォント種別が0の場合

 $8 \leq \text{X方向ボディフェイスサイズ} \leq \text{MIN}(40, \text{最大X方向ドット数})$ かつ、 $8 \leq \text{Y方向ボディフェイスサイズ} \leq \text{MIN}(40, \text{最大Y方向ドット数})$ 

注) MIN (40, 最大X方向ドット数) とは、“40” または “最大X方向ドット数” のいずれか小さい方を表します。



## フォント種別が1または2の場合

$16 \leq \text{X方向ボディフェイスサイズ} \leq \text{最大X方向ドット数}$   
 かつ、 $16 \leq \text{Y方向ボディフェイスサイズ} \leq \text{最大Y方向ドット数}$

## 注意：

X方向ボディフェイスサイズ=0またはY方向ボディフェイスサイズ=0の場合はテキストの文字フォントと同一のサイズが指定されたものとみなします。

テキストの文字フォントサイズはノーマルモードとハイレゾリレーションモードで異なるため「フォントの取得」の実行前には「フォント情報の取得」でフォントサイズを取得しフォントデータ格納サイズを確認してください。

また、この場合はテキストと同一のフォントサイズを利用するため、キャラクタフェイス開始X座標からY方向キャラクタフェイスサイズまでのパラメータは無視されます。

## ●キャラクタフェイス開始X座標

ボディフェイス内のキャラクタフェイス開始X座標を指定します。

## ●X方向キャラクタフェイスサイズ

キャラクタフェイスのX方向ドット数を指定します。

$$\left( \begin{array}{l} \text{キャラクタフェイス} \\ \text{開始X座標} \end{array} \right) + \left( \begin{array}{l} \text{X方向キャラクタ} \\ \text{フェイスサイズ} \end{array} \right) \leq \left( \begin{array}{l} \text{X方向ボディ} \\ \text{フェイスサイズ} \end{array} \right)$$

## ●キャラクタフェイス開始Y座標

ボディフェイス内のキャラクタフェイス開始Y座標を指定します。

## ●Y方向キャラクタフェイスサイズ

キャラクタフェイスのY方向ドット数を指定します。

$$\left( \begin{array}{l} \text{キャラクタフェイス} \\ \text{開始Y座標} \end{array} \right) + \left( \begin{array}{l} \text{Y方向キャラクタ} \\ \text{フェイスサイズ} \end{array} \right) \leq \left( \begin{array}{l} \text{Y方向ボディ} \\ \text{フェイスサイズ} \end{array} \right)$$

## リターン

AX = 0 ... 正常終了  
 ≠ 0 ... 異常終了

## 解説

●フォントの種別、各種フォントサイズなどはアプリケーションの起動直後では、先行のアプリケーションがこれらを変更している可能性があるため不定となります。よって、フォントドライバを利用する場合は、まず「フォント情報の設定」を実行後「フォント情報の取得」を実行してください。

- 指定されたキャラクタフェイスサイズに従い、編集の基本となるフォントが異なります。よって、指定されたキャラクタフェイスサイズと基本フォントサイズの差が大きいほど文字の歪みが大きくなります。

下記に、指定されたキャラクタフェイスサイズと基本フォントの関係を説明します。

#### 本体 CGROM のフォントを利用する場合

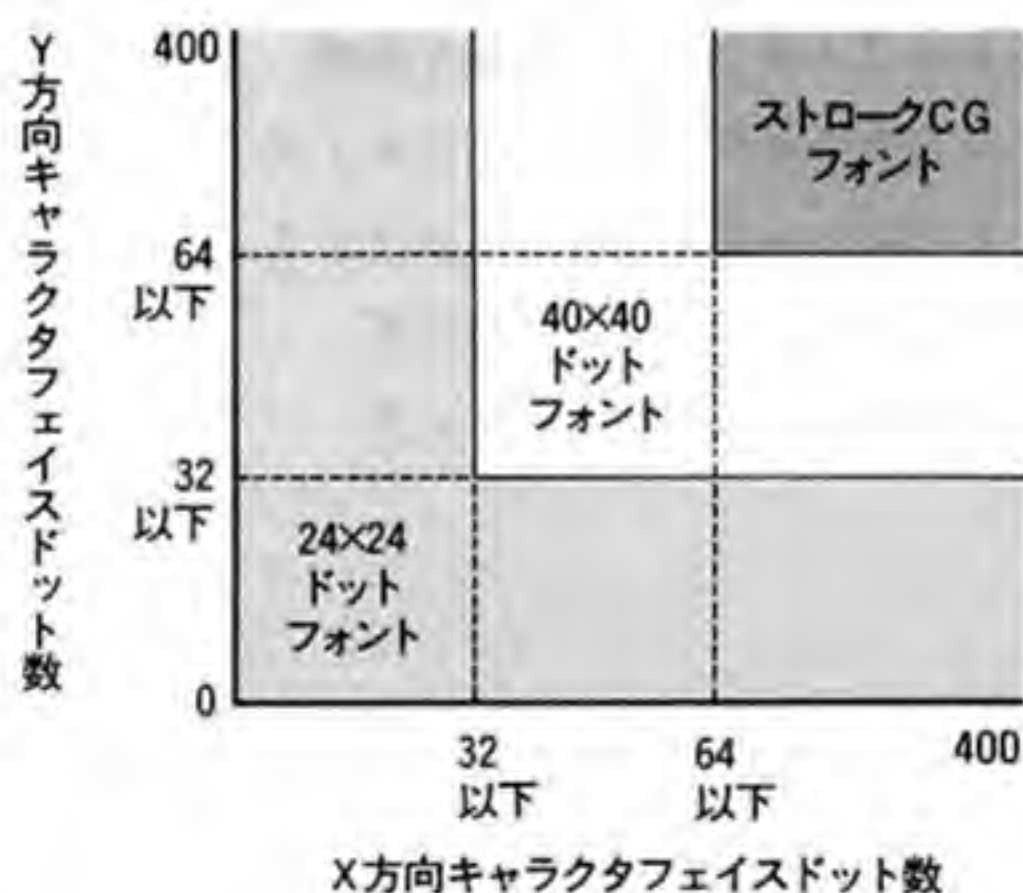
指定されたキャラクタフェイスサイズに関係なく、下記サイズのフォントを基本フォントとします。

ノーマルモード： 16×16 ドットゴシックフォント

ハイレゾリューションモード： 24×24 ドット明朝フォント

#### マルチフォント ROM ボードのフォントを利用する場合

次の図のように指定されたキャラクタフェイスサイズにより、基本フォントが異なります。

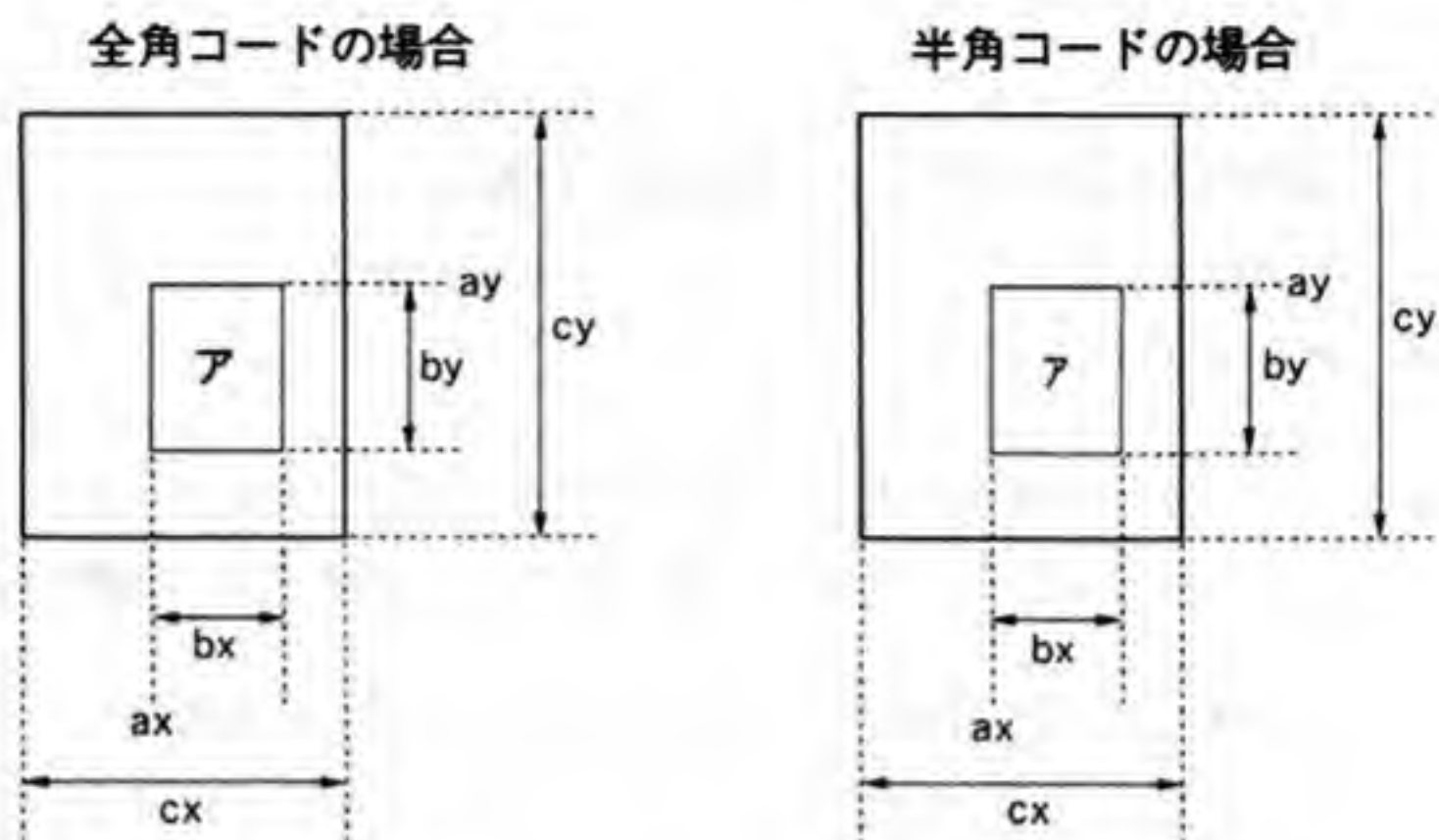


マルチフォント ROM ボード(PC-9801-38 L)の 16×16 ドットのフォントを利用する場合、指定されたキャラクタフェイスサイズに関係なく下記のサイズのフォントを基本フォントとします。

#### 8 ≤ X方向キャラクタフェイスサイズ ≤ 40 の場合の基本フォント

ノーマル/ハイレゾリューションモード共に16×16ドットフォント

- 取得する文字コードが半角コード（2921H～2B7EH）であっても，X方向ボディフェイスサイズ，Y方向ボディフェイスサイズなどのサイズは全角コードの場合と同一で字体のみ半分になります。



ax : キャラクタフェイス開始X座標  
 ay : キャラクタフェイス開始Y座標  
 bx : X方向キャラクタフェイスサイズ  
 by : Y方向キャラクタフェイスサイズ  
 cx : X方向ボディフェイスサイズ  
 cy : Y方向ボディフェイスサイズ



# **フォント情報の取得**

# **ファンクション NO. 2**

**機 能** 現在のフォント種別、フォントサイズなどの情報を通知します。

**コール** スタック = パラメータブロックの先頭アドレス

**リターン** AX = 0 … 常に正常終了

## **パラメータブロック**

| オフセット | サイズ    | 内 容             |
|-------|--------|-----------------|
| 0H    | WORD   | フォント種別          |
| 2H    | BYTE   | 横書き／縦書きフラグ      |
| 3H    | BYTE   | 動作指定フラグ         |
| 4H    | WORD   | X方向ボディフェイスサイズ   |
| 6H    | WORD   | Y方向ボディフェイスサイズ   |
| 8H    | WORD   | キャラクタフェイス開始X座標  |
| AH    | WORD   | キャラクタフェイス開始Y座標  |
| CH    | WORD   | X方向キャラクタフェイスサイズ |
| EH    | WORD   | Y方向キャラクタフェイスサイズ |
| 10H   | 16BYTE | 未使用             |

**解 説** 現在のフォント種別、フォントサイズなどの情報を取得します。フォントサイズ等については、フォント情報の設定（ファンクション NO.1）を参照してください。

## フォントの取得

## ファンクション NO. 3

フォント情報の設定(ファンクション NO.1)の動作指定フラグの設定値により、各種フォントの取得を行います。

## 1. 動作指定フラグに 0 が指定された場合

**機能** マルチフォント ROM ボード、または本体 ROM から現在のフォント情報をもとに文字フォントを取得します。(Ver1.0 互換機能)

**コール** スタック = パラメータブロックの先頭アドレス  
●パラメータブロック

| オフセット | サイズ    | 内 容                     |
|-------|--------|-------------------------|
| 0H    | WORD   | 文字コード                   |
| 2H    | WORD   | フォントデータ格納域サイズ           |
| 4H    | DWORD  | フォントデータ格納域アドレス          |
| 8H    | 24BYTE | 未使用 (常に 00 H を設定してください) |

## ●文字コード

取得する文字を 2 バイト日本語 JIS コードで指定します。2 バイト日本語 JIS コード体系外のコードが指定された場合はエラーとなります。

2 バイト日本語 JIS コード体系……2121H～7E7EH

注) マルチフォント ROM ボードからは、拡張文字(7921H～7C7EH)の取得はできません。

## ●フォントデータ格納域サイズ

バイト単位でフォントデータ格納域のサイズを指定します。格納域サイズは下記の条件を満たす必要があります。

格納域サイズ  $\geq (X \text{ 方向フォントサイズ} + 7) \div 8 \times Y \text{ 方向フォントサイズ} + 4$

注)  $\div$  は整数の割り算の商 (余り切り捨て) を意味します。

フォント情報の設定で縦書きを指定した場合、X 方向、Y 方向のフォントサイズが入れ換わることに注意してください。

## ●フォントデータ格納域アドレス

フォントデータの格納域アドレスをダブルワードで指定します。

**リターン**

AX = 0 ... 正常終了  
 ≠ 0 ... 異常終了

**解 説**

- フォントデータの格納形式を以下に説明します。

|           |                                |                                |
|-----------|--------------------------------|--------------------------------|
| 0         | X方向ボディフェイスサイズ<br>ドット数 (1 word) | Y方向ボディフェイスサイズ<br>ドット数 (1 word) |
| 4+0×α     | Y方向1ドット分のX方向mドット分のパターン(1)      |                                |
| 4+1×α     | Y方向1ドット分のX方向mドット分のパターン(2)      |                                |
| 4+2×α     | ⋮                              |                                |
| 4+n×α     | Y方向1ドット分のX方向mドット分のパターン (n)     |                                |
| 4+(n+1)×α |                                |                                |

メモリ内低位バイトから順に各ビットがX座標のドット（昇順）に対応します。

m: X方向ボディフェイスサイズ  
 n: Y方向ボディフェイスサイズ  
 α: (m+7) ¥8

(本体 ROM フォントより 16×16 ドットフォントを取得する場合の例)

**フォント情報の設定で指定するパラメータ**

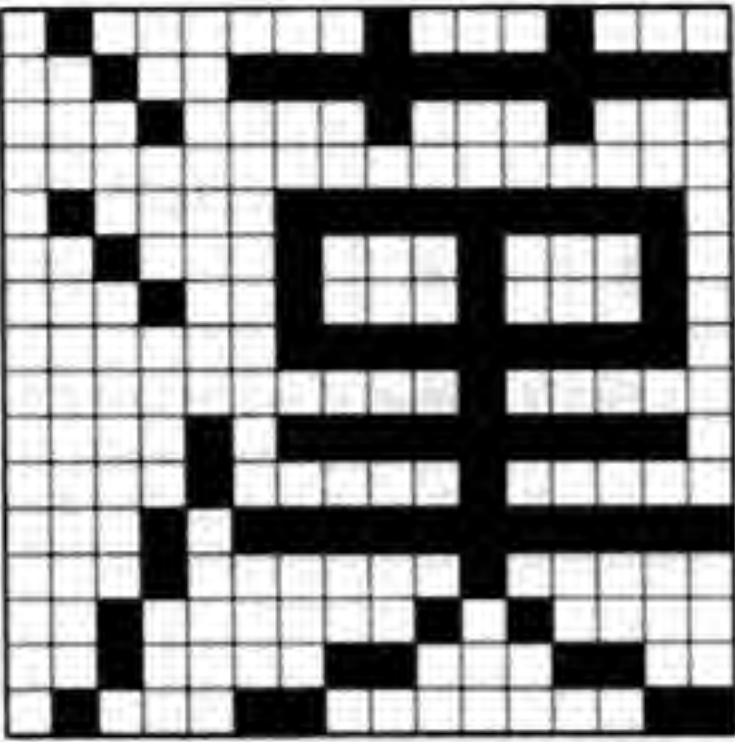
|                 |    |
|-----------------|----|
| フォント種別          | 0  |
| 横書き/縦書きフラグ      | 0  |
| 動作指定フラグ         | 0  |
| X方向ボディフェイスサイズ   | 16 |
| Y方向ボディフェイスサイズ   | 16 |
| キャラクタフェイス開始X座標  | 0  |
| キャラクタフェイス開始Y座標  | 0  |
| X方向キャラクタフェイスサイズ | 16 |
| Y方向キャラクタフェイスサイズ | 16 |

**フォントの取得で設定するパラメータ**

|                |         |
|----------------|---------|
| 文字コード          | 3441 H  |
| フォントデータ格納域サイズ  | 36      |
| フォントデータ格納域アドレス | 任意のアドレス |



次に示すフォントイメージの場合、フォントデータは次のようになります。

| フォントイメージ                                                                            | フォントデータ          |
|-------------------------------------------------------------------------------------|------------------|
|  | ← WORD →         |
|                                                                                     | 0010 H 0 X方向ドット数 |
|                                                                                     | 0010 H 2 Y方向ドット数 |
|                                                                                     | 8840 H 4         |
|                                                                                     | FF27 H 6         |
|                                                                                     | 8810 H 8         |
|                                                                                     | 0000 H 10        |
|                                                                                     | ⋮                |
|                                                                                     | 5020 H 30        |
|                                                                                     | 8C21 H 32        |
|                                                                                     | 0346 H 34        |
|                                                                                     |                  |

注) 動作フラグの指定で0を指定すると、Ver 1.0の互換動作となるため、CONFIG.SYSファイルで/Mスイッチをどのように指定しても縦横それぞれ128ドットを越えるフォントの取得は行えません。

## 2. 動作指定フラグに1が指定された場合

**機能** 現在のフォント情報を基に文字フォントを取得します。取得可能なフォントサイズが8×8～400×400ドットであること、フォントの取得に必要な作業域を利用者が確保する以外、機能は＜動作指定フラグに0が指定された場合＞と同じです。

**コール** スタック = パラメータブロックの先頭アドレス

### ●パラメータブロック

| オフセット | サイズ    | 内 容                 |
|-------|--------|---------------------|
| 00H   | WORD   | 文字コード               |
| 02H   | WORD   | フォントデータ格納域サイズ       |
| 04H   | DWORD  | フォントデータ格納域アドレス      |
| 08H   | WORD   | 作業域サイズ              |
| 0AH   | DWORD  | 作業域アドレス             |
| 0EH   | 18BYTE | 未使用（常に00Hを設定してください） |

### ●作業域サイズ

フォント取得のために必要な作業域を確保し、そのサイズを設定します。  
必要な作業域のサイズは取得するフォントのサイズから以下のように算出します。

$$\text{作業域サイズ} \geq (A+7) \times 8 \times A + 15$$

注)  $\times$ マークは切捨ての整数系の割り算を意味します。

$A = X, Y$  方向キャラクタフェイスサイズの値の大きい方

例: 60×70 ドットフォントを取得するときのサイズ

$$\text{作業域サイズ} \geq 645 = (70+7) \times 8 \times 70 + 15$$

・ 16×16, 24×24, 40×40 ドットフォントから基本フォントを取得する場合。

$$\text{作業域サイズ} \geq W1 + W2$$

注)  $\times$ マークは切捨ての整数系の割り算を意味します。

$$W1 = (B+7) \times 8 \times B$$

$$W2 = A \times 8 \times B$$

$A$  = 基本フォントサイズ (40, 24, 16)

$B = A$  または,  $X, Y$  方向キャラクタフェイスサイズの値の大きい方

例: 30×45 ドットフォントを取得するときのサイズ

$$\text{作業域サイズ} \geq 405 = (45+7) \times 8 \times 45 + 24 \times 8 \times 45$$

### ●作業域アドレス

利用者が確保した作業域の先頭をダブルワードで指定します。

## 3. 動作指定フラグに 2 が指定された場合

**機 能**     利用者が定義したフォントイメージの拡大/縮小を行います。

**コール**     スタック = パラメータブロックの先頭アドレス

●パラメータブロック

| オフセット | サイズ    | 内 容                   |
|-------|--------|-----------------------|
| 00H   | DWORD  | 利用者定義フォントアドレス         |
| 04H   | WORD   | X方入力キャラクタフェイスサイズ      |
| 06H   | WORD   | Y方向入力キャラクタフェイスサイズ     |
| 08H   | WORD   | フォントデータ格納域サイズ         |
| 0AH   | DWORD  | フォントデータ格納域アドレス        |
| 0EH   | WORD   | 作業域サイズ                |
| 10H   | DWORD  | 作業域アドレス               |
| 14H   | 12BYTE | 未使用（常に 00H を設定してください） |

●利用者定義フォントアドレス

ダブルワードで指定した利用者定義フォントアドレスには、以下のフォーマットでフォントが格納されていなければなりません。

|                       |                           |                                                                                                                        |
|-----------------------|---------------------------|------------------------------------------------------------------------------------------------------------------------|
| $0 \times \alpha$     | Y方向1ドット分のX方向mドット分のパターン(1) | メモリ内低位バイト, MSB 側から順に各ビットが X 座標のドット (昇順) に対応します。<br><br>m   : X方向入力ドット数<br>n   : Y方向入力ドット数<br>$\alpha$ : $(m+7) \div 8$ |
| $1 \times \alpha$     | Y方向1ドット分のX方向mドット分のパターン(2) |                                                                                                                        |
| $2 \times \alpha$     | ⋮                         |                                                                                                                        |
| $n \times \alpha$     | Y方向1ドット分のX方向mドット分のパターン(n) |                                                                                                                        |
| $(n+1) \times \alpha$ |                           |                                                                                                                        |

●X方向, Y方向入力キャラクタフェイスサイズ

入力するフォントサイズは  $16 \times 16 \sim 64 \times 64$  の範囲でなければなりません。

●フォントデータ格納域サイズ

バイト単位でフォントデータ格納域のサイズを指定します。格納域サイズは下記の条件を満たさなければなりません。

格納域サイズ  $\geq (X \text{ 方向ボディフェイスサイズ} + 7) \div 8 \times Y \text{ 方向ボディフェイスサイズ} + 4$

注)  $\div$  マークは余り切捨ての整数系の割り算を意味します。



### ●フォントデータ格納域アドレス

フォントデータの格納域アドレスの先頭をダブルワードで指定します。

指定した格納アドレスには、フォントの取得で動作指定フラグに0または1を指定した場合と同じ形式で拡大/縮小されたフォントデータが格納されます。

### ●作業域サイズ

バイト単位でフォント取得のために必要な作業域を確保し、そのサイズを指定しなければなりません。

作業域サイズは下記の条件を満たす必要があります。

**作業域サイズ $\geq$ W1+W2**

注) ¥マークは切捨ての整数系の割り算を意味します。

$$W1 = (A + 7) \text{ ¥ } 8 \times A$$

$$W2 = (Y \text{ 方向キャラクタフェイスサイズ} + 7) \text{ ¥ } 8 \times A$$

A=X, Y 方向入力キャラクタフェイスサイズ, X, Y 方向キャラクタフェイスサイズの値の大きい方

例: 24×24 ドットの利用者定義フォントから、30×40 ドットフォントを取得するときの作業域サイズ

$$\text{作業域サイズ} \geq 320 = (40 + 7) \text{ ¥ } 8 \times 40 + (24 + 7) \text{ ¥ } 8 \times 40$$

### ●作業域アドレス

作業域の先頭アドレスをダブルワードで指定します。

## 6.5 エラーコード一覧

ここでは、エラーコードについて説明します。

各ファンクションでは、通常、リターン値が AX≠0 のときはエラーコードを示します。以下にエラーコードとその意味について説明します。

| エラーコード | 意 味                                                 |
|--------|-----------------------------------------------------|
| 01     | 不正呼び出し。パラメータに誤りがあります。                               |
| 02     | CONFIG.SYSファイルで定義した上限(最大ボディフェイスサイズ)を超えているため実行不可能です。 |
| 03     | 現在のハードウェアではこの機能は利用できません。                            |
| 04     | 指定された文字コードは 2 バイト JISコード体系です。                       |
| 05     | 利用環境の設定により、ストローク CG フォントは利用できません。                   |



# 索引

## 本体機能

### A~Z

|                       |       |
|-----------------------|-------|
| CTRL+FUNC キー .....    | 12    |
| RS-232C .....         | 2, 10 |
| RS-232C ポートの初期化 ..... | 2     |
| RS-232C ポートの操作 .....  | 10    |

### あ〜ん

|                 |      |
|-----------------|------|
| 拡張機能 .....      | 1    |
| キーの取得 .....     | 4    |
| キーの設定 .....     | 8    |
| 受信データ長 .....    | 10   |
| 直接コンソール出力 ..... | 13   |
| ファンクションキー ..... | 4, 8 |
| プリンタモード .....   | 17   |

## 日本語処理

### 数字

|                              |    |
|------------------------------|----|
| 1 バイト JIS 文字列を全角文字列に変換 ..... | 28 |
| 1 バイト JIS 文字列を半角文字列に変換 ..... | 29 |
| 2 バイト JIS をシフト JIS に変換 ..... | 45 |

### A~Z

|                   |    |
|-------------------|----|
| AI かな漢字変換 .....   | 19 |
| EMS .....         | 21 |
| NECAI.SYS .....   | 21 |
| NECAIK1.DRV ..... | 19 |
| NECAIK2.DRV ..... | 19 |
| NECDIC.DRV .....  | 19 |
| NECDIC.SYS .....  | 20 |

### あ

|                            |        |
|----------------------------|--------|
| アプリケーションプログラム .....        | 23, 24 |
| アプリケーションプログラムからの使用禁止 ..... | 24     |
| アプリケーションプログラムへの開放 .....    | 23     |

### か

|             |        |
|-------------|--------|
| カナ文字列 ..... | 27     |
| 学習 .....    | 35, 64 |

|                           |    |
|---------------------------|----|
| 学習（連文節） .....             | 64 |
| 学習機能の有無 .....             | 26 |
| キーボードからの日本語入力の禁止・許可 ..... | 25 |
| 語句の学習 .....               | 35 |
| 語句の削除 .....               | 34 |
| 語句の登録 .....               | 33 |
| 語句の変換（文節変換：最初の候補） .....   | 37 |
| 語句の変換（文節変換：次候補） .....     | 39 |
| 語句の変換（文節変換：前候補） .....     | 40 |

### さ

|                               |        |
|-------------------------------|--------|
| 先読み機能 .....                   | 67     |
| シフト JIS を 2 バイト JIS に変換 ..... | 46     |
| 次候補（連文節変換） .....              | 59     |
| 辞書のオープン .....                 | 30     |
| 辞書のクローズ .....                 | 32     |
| 辞書の先読み .....                  | 50     |
| 辞書の先読みと逐次変換 .....             | 50     |
| 先読み機能の有無 .....                | 67     |
| 辞書ファイル名 .....                 | 19, 20 |
| 全角文字列 .....                   | 28     |
| 前候補（連文節変換） .....              | 62     |



## た

逐次変換ドライバの有無の取得 .....47

## な

日本語処理 .....19

日本語入力モード .....41,42

日本語入力モードから抜ける .....42

日本語入力モードに入る .....41

日本語入力のモード取得 .....44

日本語入力のモードセット .....43

日本語入力用デバイスドライバ .....19

## は

半角文字列 .....29

文節変換 .....19

変換ドライバ .....21

## ら

連文節変換 .....56

連文節変換（次候補） .....59

連文節変換（前候補） .....62

ローマ字をカナ文字列に変換 .....27

## マウス

## A～Z

MOUSE.SYS .....69

## あ

移動距離 .....85,88

移動範囲 .....89,90

押下情報 .....79,81

## か

カーソル位置の取得 .....77

カーソル位置の設定 .....78

カーソル移動範囲 .....89,90

カーソル消去 .....76

カーソルの形の設定 .....83

カーソルの表示画面の設定 .....91

カーソル表示 .....75

解放情報 .....80,82

環境のチェック .....74

グラフィック用 VRAM の設定と実装状況の取得  
..... 92

## さ

座標系 .....71

初期値 .....93

垂直方向のカーソル移動範囲の設定 .....90

水平方向のカーソル移動範囲の設定 .....89

## た

中心点 .....83

## は

左ボタンの押下情報の取得 .....79

左ボタンの解放情報の取得 .....80

表示画面 .....72

## ま

マウス .....69

マウスインターフェイス .....69

マウスカーソル .....73

マウスカーソルの形状 .....83

マウスの移動距離 .....85,88

マウスの移動距離の取得 .....85

マウス用デバイスドライバ .....69

右ボタンの押下情報の取得 .....81

右ボタンの解放情報の取得 .....82

ミッキー .....73,88

ミッキー／ドット比の設定 .....88

## や

ユーザー定義サブルーチンのコール条件の設定  
.....86

## わ

割り込み周期 .....71

割り込みベクタ .....71

## グラフィックス

## A～Z

|                 |    |
|-----------------|----|
| GRAPH.LIB ..... | 95 |
| GRAPH.SYS ..... | 95 |

## あ

|                |     |
|----------------|-----|
| 裏返し .....      | 148 |
| エラーコード .....   | 165 |
| エントリテーブル ..... | 97  |
| 円の描画 .....     | 137 |

## か

|                       |     |
|-----------------------|-----|
| 回転 .....              | 148 |
| 拡大 .....              | 147 |
| 仮想 VRAM の生成 .....     | 106 |
| 画面消去 .....            | 125 |
| カラーコード .....          | 113 |
| グラフィックイメージの取得 .....   | 143 |
| グラフィックイメージの設定 .....   | 145 |
| グラフィックスライブラリ .....    | 95  |
| グラフィックの開始 .....       | 104 |
| グラフィックの終了 .....       | 105 |
| グラフィック用デバイスドライバ ..... | 95  |

## さ

|                    |          |
|--------------------|----------|
| 三角形の描画 .....       | 130      |
| 指定座標のパレットの取得 ..... | 162      |
| 縮小 .....           | 147      |
| 使用例 .....          | 100, 102 |
| 初期設定 .....         | 104      |
| 線の描画 .....         | 128      |

## た

|                   |     |
|-------------------|-----|
| 台形の描画 .....       | 135 |
| 楕円の描画 .....       | 139 |
| 中断処理ルーチンの取得 ..... | 164 |
| 中断処理ルーチンの設定 ..... | 123 |
| 長方形の描画 .....      | 133 |
| 点の描画 .....        | 126 |

## な

|             |     |
|-------------|-----|
| 塗りつぶし ..... | 141 |
|-------------|-----|

## は

|                      |     |
|----------------------|-----|
| バージョンの取得 .....       | 150 |
| バックグラウンドカラーの取得 ..... | 159 |
| バックグラウンドカラーの設定 ..... | 119 |
| パレットの取得 .....        | 156 |
| パレット番号の設定 .....      | 113 |
| 表示スイッチの取得 .....      | 161 |
| 表示スイッチの設定 .....      | 121 |
| 表示プレーンの設定 .....      | 112 |
| 表示モードの取得 .....       | 153 |
| 表示モードの設定 .....       | 108 |
| 表示領域の取得 .....        | 163 |
| 表示領域の設定 .....        | 122 |
| ビューポート領域の取得 .....    | 157 |
| ビューポート領域の設定 .....    | 116 |
| 描画プレーンの取得 .....      | 154 |
| 描画プレーンの設定 .....      | 110 |
| フォアグラウンドカラーの取得 ..... | 158 |
| フォアグラウンドカラーの設定 ..... | 118 |
| プレーン数の取得 .....       | 151 |
| 閉領域の塗りつぶし .....      | 141 |
| ボーダーカラーの取得 .....     | 160 |
| ボーダーカラーの設定 .....     | 120 |

## ら

|            |     |
|------------|-----|
| 領域移動 ..... | 149 |
| 領域転送 ..... | 147 |



## EMSインターフェイス

## A~Z

|                                             |     |
|---------------------------------------------|-----|
| Allocate Alternate Map Register Set         | 273 |
| Allocate DMA Register Set                   | 277 |
| Allocate Pages                              | 184 |
| Allocate Raw Pages                          | 260 |
| Allocate Standard Pages                     | 257 |
| Alter Page Map & Call                       | 234 |
| Alter Page Map & Jump                       | 231 |
| Alternate Map Register Set                  | 264 |
| Deallocate Alternate Map Register Set       | 275 |
| Deallocate DMA Register Set                 | 283 |
| Deallocate Pages                            | 189 |
| Disable DMA on Alternate Map Register Set   | 281 |
| Disable OS/E Function Set                   | 288 |
| DMA レジスタセットのアロケート                           | 277 |
| DMA レジスタセットの解放                              | 283 |
| Enable/Disable Page frame                   | 296 |
| Enable DMA on Alternate Map Register Set    | 279 |
| Enable OS/E Function Set                    | 285 |
| Exchange Memory Region                      | 244 |
| Get & Set Page Map                          | 203 |
| Get All Handle Pages                        | 199 |
| Get Alternate Map Register Set              | 266 |
| Get Alternate Map Save Array Size           | 272 |
| Get Attribute Capability                    | 222 |
| Get Handle Attribute                        | 218 |
| Get Handle Count                            | 197 |
| Get Handle Directory                        | 227 |
| Get Handle Name                             | 223 |
| Get Handle Pages                            | 198 |
| Get Hardware Configuration Array            | 252 |
| Get interrupt vector 法                      | 309 |
| Get Mappable Physical Address Array         | 249 |
| Get Mappable Physical Address Array Entries | 251 |
| Get Page Frame Address                      | 182 |

|                                                |          |
|------------------------------------------------|----------|
| Get Page Map                                   | 201      |
| Get Page Map Stack Size                        | 238      |
| Get Pageframe Status                           | 295      |
| Get Partial Page Map                           | 206      |
| Get Size of Page Map Save Array                | 205      |
| Get Size of Partial Page Map Save Array        | 209      |
| Get Status                                     | 181      |
| Get Total Handles                              | 230      |
| Get Unallocated Page Count                     | 183      |
| Get Unallocated Raw Page Count                 | 255      |
| Get Version                                    | 191      |
| Map/Unmap Handle Page                          | 187      |
| Map/Unmap Multiple Handle Pages                | 210, 212 |
| Move Memory Region                             | 239      |
| Open handle 法                                  | 305      |
| OS ファンクションセットの使用許可                             | 285      |
| OS/E ファンクションセットの使用禁止状態の設定                      | 288      |
| Prepare Expanded Memory Hardware for Warm Boot | 284      |
| Reallocate Pages                               | 215      |
| Restore Page Map                               | 194      |
| Return Access Key                              | 290      |
| Save Page Map                                  | 192      |
| Search for Named Handle                        | 229      |
| Set Alternate Map Register Set                 | 269      |
| Set Handle Attribute                           | 220      |
| Set Handle Name                                | 225      |
| Set Page Map                                   | 202      |
| Set Partial Page Map                           | 208      |
| TSR プログラム                                      | 176      |

## あ

|             |     |
|-------------|-----|
| アクセスキーのリターン | 288 |
| アロケート       | 184 |
| アンマッピング     | 171 |
| インプリメンテーション | 311 |
| ウィンドウ       | 167 |



- ウォームブート .....284
- 応用ファンクション .....173
- 応用プログラミング .....171
- オペレーティングシステム .....173, 175, 313
  
- か
- 拡張メモリ .....167
- 拡張メモリの量 .....311
- 拡張メモリマネージャ .....167
- 基本ファンクション .....170
- ケイパビリティ .....222
  
- さ
- 再アロケート .....312
- システムローハンドル .....308
- ステータスコードのクロスリファレンス 297, 300
- ステータスの取得 .....181
- 全ハンドルページの取得 .....199
  
- た
- 代替マップセーブ配列のサイズ取得 .....272
- 代替マップレジスタ上のDMAの使用許可 279
- 代替マップレジスタ上のDMAの使用不許可  
.....281
- 代替マップレジスタセットのアロケート ...273
- 代替マップレジスタセットのセット .....269
- 代替マップレジスタセットの解放 .....275
- 代替マップレジスタセットの取得 .....266
- デアロケート .....189
- ディレクトリ .....227
  
- な
- 名前の割り当て .....172
  
- は
- ハードウェア構成配列の取得 .....252
- 配列のサイズ取得 .....209
- ハンドル .....311
- ハンドルアトリビュートのケイパビリティ 222
- ハンドルアトリビュートの取得 .....218
- ハンドルアトリビュートのセット .....220
- ハンドルカウンタの取得 .....197
- ハンドル数 .....311
- ハンドルの検索 .....171
- ハンドルのサーチ .....229
- ハンドルの総数 .....230
- ハンドルの属性 .....172
- ハンドルのディレクトリの取得 .....227
- ハンドルの番号づけ .....311
- ハンドルページ .....187
- ハンドルページの取得 .....198
- ハンドル名の取得 .....223
- ハンドル名のセット .....225
- バージョンの取得 .....191
- 標準サイズのページのアロケート .....257
- 複数ハンドルページのマッピング／物理ページ  
の解除 .....210, 212
- 物理アドレス配列 .....249
- 物理アドレス配列エントリ .....251
- ページカウンタ .....171
- ページのアロケート .....184
- ページの再アロケート .....215
- ページフレーム .....167
- ページフレームアドレス .....182
- ページフレームの管理 .....292
- ページフレーム用バンクのステータスを取得  
.....295
- ページフレーム用バンクの状態の設定 .....296
- ページマッピングの変更とコール .....234
- ページマッピングの変更とジャンプ .....231
- ページマップ .....173
- ページマップスタックサイズの取得 .....238
- ページマップセーブ配列のサイズ取得 .....205
- ページマップの取得 .....201
- ページマップの取得とセット .....203
- ページマップのセーブ .....192
- ページマップのセット .....202
- ページマップのリストア .....194
  
- ま
- マッピング .....171
- マッピングコンテキスト .....188, 317

|                               |     |
|-------------------------------|-----|
| マッピングコンテキストのセーブ .....         | 206 |
| マッピングコンテキストのリストア .....        | 208 |
| マップ可能な物理アドレス配列の取得 .....       | 249 |
| マッピング可能な物理アドレス配列エントリの取得 ..... | 251 |
| マップ可能メモリ領域 .....              | 206 |
| マップレジスタの変更 .....              | 264 |
| 未アロケートページカウント .....           | 183 |
| 未アロケートのローページカウントの取得 .....     | 255 |
| メモリ量 .....                    | 173 |

|                   |     |
|-------------------|-----|
| メモリ領域の移動 .....    | 239 |
| メモリ領域の移動と交換 ..... | 173 |
| メモリ領域の交換 .....    | 244 |

## ら

|                   |     |
|-------------------|-----|
| リアロケート .....      | 172 |
| ローハンドル .....      | 311 |
| ローページのアロケート ..... | 260 |
| 論理ページ .....       | 167 |

## フォントドライバ

## A~Z

|                        |     |
|------------------------|-----|
| FONT. SYS .....        | 320 |
| X 方向キャラクタフェイスサイズ ..... | 328 |
| X 方向ボディフェイスサイズ .....   | 327 |
| Y 方向キャラクタフェイスサイズ ..... | 328 |
| Y 方向ボディフェイスサイズ .....   | 327 |

## あ

|                |     |
|----------------|-----|
| エラーコード一覧 ..... | 337 |
| エントリテーブル ..... | 321 |

## か

|                        |          |
|------------------------|----------|
| キャラクタフェイス開始 X 座標 ..... | 328      |
| キャラクタフェイス開始 Y 座標 ..... | 328      |
| グラフィック VRAM .....      | 319, 324 |
| グラフィックイメージの設定 .....    | 324      |
| グラフィックイメージデータ .....    | 324      |
| グラフィックスドライバ .....      | 319, 324 |

## さ

|                          |     |
|--------------------------|-----|
| 最小ボディフェイスサイズ .....       | 320 |
| 最大ボディフェイスサイズ .....       | 320 |
| スイッチ (/M, /E スイッチ) ..... | 320 |
| 先頭アドレスの取得 .....          | 321 |

## た

|               |     |
|---------------|-----|
| 動作指定フラグ ..... | 327 |
|---------------|-----|

## は

|                      |          |
|----------------------|----------|
| バージョンの取得 .....       | 325      |
| パラメータブロック .....      | 321      |
| ファンクション一覧 .....      | 321      |
| ファンクションの呼び出し .....   | 321      |
| フォント種別 .....         | 326      |
| フォント情報の取得 .....      | 331      |
| フォント情報の設定 .....      | 326      |
| フォントデータ格納域アドレス ..... | 332      |
| フォントデータ格納域サイズ .....  | 332      |
| フォントドライバ .....       | 319, 320 |
| フォントドライバの呼び出し .....  | 321      |
| フォントドライバの組み込み .....  | 317      |
| フォントの取得 .....        | 332      |

## ま

|                       |     |
|-----------------------|-----|
| マルチフォント ROM ボード ..... | 319 |
| 文字コード .....           | 332 |

## や

|                     |     |
|---------------------|-----|
| 横書き/縦書きフラグ .....    | 323 |
| 利用者定義フォントアドレス ..... | 336 |

## 記号

|               |     |
|---------------|-----|
| /E スイッチ ..... | 320 |
| /M スイッチ ..... | 320 |









**NEC**

